Magellan® Motion Processor
Programmer's Command Reference

Performance Motion Devices, Inc.
55 Old Bedford Road
Lincoln, MA 01773

Revision 2.5, September 2008

NOTICE

This document contains proprietary and confidential information of Performance Motion Devices, Inc., and is protected by federal copyright law. The contents of this document may not be disclosed to third parties, translated, copied, or duplicated in any form, in whole or in part, without the express written permission of PMD.

The information contained in this document is subject to change without notice. No part of this document may be reproduced or transmitted in any form, by any means, electronic or mechanical, for any purpose, without the express written permission of PMD.

Copyright 1998–2008 by Performance Motion Devices, Inc.

Magellan, ION, Magellan/ION, Pro-Motion, C-Motion, and VB-Motion are trademarks of Performance Motion Devices, Inc.

Warranty

PMD warrants performance of its products to the specifications applicable at the time of sale in accordance with PMD's standard warranty. Testing and other quality control techniques are utilized to the extent PMD deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Performance Motion Devices, Inc. (PMD) reserves the right to make changes to its products or to discontinue any product or service without notice, and advises customers to obtain the latest version of relevant information to verify, before placing orders, that information being relied on is current and complete. All products are sold subject to the terms and conditions of sale supplied at the time of order acknowledgement, including those pertaining to warranty, patent infringement, and limitation of liability.

Safety Notice

Certain applications using semiconductor products may involve potential risks of death, personal injury, or severe property or environmental damage. These products are not designed, authorized, or warranted to be suitable for use in life support devices or systems or other critical applications. Inclusion of PMD products in such applications is understood to be fully at the customer's risk.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards must be provided by the customer to minimize inherent procedural hazards.

Disclaimer

PMD assumes no liability for applications assistance or customer product design. PMD does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of PMD covering or relating to any combination, machine, or process in which such products or services might be or are used. PMD's publication of information regarding any third party's products or services does not constitute PMD's approval, warranty, or endorsement thereof.

Related Documents

Magellan Motion Processor User's Guide

Complete description of the Magellan Motion Processor features and functions with detailed theory of its operation.

Magellan Motion Processor Electrical Specifications

Booklets containing physical and electrical characteristics, timing diagrams, pinouts, and pin descriptions of each series:

MC58000 Series, for DC brush, brushless DC, Microstepping, and Pulse & Direction motion processors

MC55000 Series, for Pulse & Direction motion processors

Magellan Motion Processor Developer's Kit Manual

How to install and configure the DK58000 series and DK55000 series developer's kit PC board.

Pro-Motion User's Guide

User's guide to Pro-Motion, the easy-to-use motion system development tool and performance optimizer. Pro-Motion is a sophisticated, easy-to-use program which allows all motion parameters to be set and/or viewed, and allows all features to be exercised.

Other Documents

ION Digital Drive User's Manual

How to install and configure ION Digital Drives.

Prodigy-PCI Motion Card User's Guide

How to install and configure the Prodigy-PCI motion board.

Prodigy-PC/104 Motion Card User's Guide

How to install and configure the Prodigy-PC/104 motion board.

Table of Contents

1. The Magellan Family 7
1.1 Family Summary 7
1.2 Magellan Motion Processor Products 8

2. C-Motion 11
2.1 Introduction 11
2.2 Files 11
2.3 Using C-Motion 12
2.4 Prodigy Motion Card Specific Commands 14

3. VB-Motion 15
3.1 Introduction 15
3.2 Files 15
3.3 Using VB-Motion 16
3.4 Prodigy Motion Card Specific Commands 17

4. Instruction Reference 19
4.1 How to Use This Reference 19

5. Instruction Summary Tables 171
5.1 Descriptions by Functional Category 171
5.2 Alphabetical Listing 174
5.3 Numerical Listing 177
5.4 Magellan Compatibility 179

This page intentionally left blank.

1. The Magellan Family

This manual provides a Programmer's Command Reference for the Magellan® Family of Motion Processors from PMD including the MC58000 Series (DC brush, brushless DC, microstepping, and step motor), the MC55000 Series (pulse & direction step motor) motion processors, and the Magellan® /ION® motion processor. In addition, Magellan processors are used in a number of card-level products including the Prodigy-PCI and Prodigy-PC/104 motion cards. If you are using one of these card or module-level products, the exact motion processor type can be determined from the corresponding User's Manual.

Each Magellan is a complete chip-based motion processor, providing trajectory generation and related motion control functions. Depending on the type of motor to be controlled, it provides servo loop closure, on-board commutation for brushless motors, and high-speed pulse & direction outputs. Together, these products provide a software-compatible family of dedicated motion processors that can handle a large variety of system configurations.

Each of the multi-chip versions of these products utilizes a similar architecture, consisting of a high-speed computation unit along with an ASIC (Application Specific Integrated Circuit). The computation unit contains special on-board hardware, which makes it well suited for the task of motion control. Single axis/single chip configurations of Magellan are also available, in which case, the logic provided in the ASIC is integrated directly with the high speed computation unit.

Along with similar hardware architecture, these chips also share most software commands. Therefore, software written for one motion processor may be re-used with the other independent of motor type or hardware configuration.

1.1 Family Summary

The various members of the Magellan family are designed for differing motor types and applications:

MC58000 Series (MC58420, MC58320, MC58220, MC58120, MC58110)—This series supports DC brush, brushless DC, and step motors using both pulse & direction and microstepping output formats. For use with DC brush or brushless DC with external commutation it outputs in PWM or DAC-compatible format. For use with two-phase or three-phase brushless DC motors it outputs in PWM or DAC-compatible format. For use with pulse & direction step motors it outputs in pulse & direction format, and for use with microstepping step motors it outputs PWM or DAC-compatible formats.

MC55000 Series (MC55420, MC55320, MC55220, MC55120, MC55110)—This series outputs pulse & direction signals for use with step motors.

Magellan/ION—This single-chip motion processor is specifically designed to work with the ION family of digital drives. It provides one axis of control, with an additional auxiliary axis of encoder input. It controls either a DC brush motor, a three-phase brushless motor, or a step motor. Compared to the MC50000, it has additional amplifier control features such as digital current control and overtemperature sense. The Magellan/ION is only available embedded in the ION Digital Drive; it is not sold as a separate motion processor device.

1.2 Magellan Motion Processor Products

The following table presents a feature summary of the products in the Magellan Motion Processor product family.

	MC58000 Series	MC55000 Series	Magellan/ION
# of axes	1, 2, 3, 4	1, 2, 3, 4	1
Motor types supported	DC brush, brushless DC, Microstepping step motor, Pulse & Direction step motor	Pulse & Direction step motor	DC brush, brushless DC, Microstepping step motor
Microstepping motor			
Output format	PWM, DAC, Pulse & Direction	Pulse & Direction	PWM (internal to drive)
Parallel communication	✓	✓	
Serial communication	✓	✓	✓
CAN 2.0B communication	✓	✓	✓
Incremental encoder input	✓	✓	✓
Parallel word device input	✓	✓	
Index & Home signals	✓	✓	✓
Position capture	✓	✓	✓
Directional limit switches	✓	✓	✓
PWM output	✓		amplifier is internal
Parallel DAC output	✓		
SPI DAC output	✓		
Pulse & direction output	✓	✓	
Digital current control			✓
Field oriented control			✓
Under/overvoltage sense			✓
Current foldback			✓
Trapezoidal profiling	✓	✓	✓
Velocity profiling	✓	✓	✓
S-curve profiling	✓	✓	✓
Electronic gearing	✓	✓	✓
On-the-fly changes	✓	✓	✓
PID position servo loop	✓	✓	✓
Dual biquad filters	✓		✓
Dual encoder loop	✓ (multi-axis configurations only)		✓
Programmable derivative sampling time	✓		✓
Feedforward (accel & vel)	✓		✓
Data trace/diagnostics	✓	✓	✓
Motion error detection	✓	✓ (requires encoder)	✓
Axis settled indicator	✓	✓ (requires encoder)	✓
Analog input	✓	✓	
Programmable bit output	✓	✓	✓
Software-invertible signals	✓	✓	✓
User-defined I/O	✓	✓	
External RAM support	✓	✓	
Multi-chip synchronization	✓		

	MC58000 Series	MC55000 Series	Magellan/ION
Chipset configurations	MC58420 (4 axes, 2 ICs) MC58320 (3 axes, 2 ICs) MC58220 (2 axes, 2 ICs) MC58120 (1 axis, 2 ICs) MC58110 (1 axis, 1 IC)	MC55420 (4 axes, 2 ICs) MC55320 (3 axes, 2 ICs) MC55220 (2 axes, 2 ICs) MC55120 (1 axis, 2 ICs) MC55110 (1 axis, 1 IC)	Magellan/ION sold as part of ION drive only
Motion processor developer's kit p/n's	DK58420 (4 axes, 2 ICs) DK58110 (1axis, 1 IC)	DK55420 (4 axes, 2 ICs) DK55110 (1 axis, 1 IC)	Not available as motion processor developer's kit. Used exclusively in ION products

This page intentionally left blank.

2. C-Motion

2.1 Introduction

C-Motion is a "C" source code library that contains all the code required for communicating with the Magellan Motion Processor.

C-Motion includes the following features:

- Axis virtualization.
- The ability to communicate to multiple Magellan Motion Processors.
- Can be easily linked to any "C/C++" application.

C-Motion callable functions are broken into two groups, those callable functions that encapsulate motion processor specific commands, and those callable functions that encapsulate product-specific capabilities.

The motion processor specific commands are detailed in Chapter 4, *Instruction Reference*. They are the primary commands that you will use to control the major motion features including profile generation, servo loop closure, motor output signal generation (PWM and analog), breakpoint processing, trace operations, and many other functions.

Each Magellan Motion Processor command has a C-Motion command of the identical name, but prefaced by the letters "PMD." For example, the Magellan command **SetPosition** is called **PMDSetPosition**.

2.2 Files

The following table lists the files that make up the C-Motion distribution.

C-Motion.h/C-Motion.c	Definition/declaration of the PMD Magellan command set
PMDpar.h/PMDpar.c	Parallel interface functions
PMDW32ser.h/PMDW32ser.c	Windows serial communication interface functions
PMDdrv.h/PMDdrv.c	Windows ISA driver communication interface functions
PMDpci.h/PMDpci.c	Windows PCI driver communication interface functions*
PMDutil.h/PMDutil.c	General utility functions
PMDtrans.h/PMDtrans.c	Generic transport (interface) functions
PMDecode.h	Defines the PMD Magellan and C-Motion error codes
PMDocode.h	Defines the control codes for Magellan commands
PMDtypes.h	Defines the basic types required by C-Motion
PMDdiag.h	Defines a string structure for each command code that can be used for debugging or diagnostics

**The PMDpci.c file cannot be compiled without the PLX source which can be obtained separately from PLX Technology. In order to build a PCI program without the PLX source, the PMDpci.obj and PlxApi.lib files must be linked in. These files are located in the \Debug and \Release subdirectories.*

2.3 Using C-Motion

C-Motion can be linked to your application code by including the above "C" source files in your application. Then, for any application source file that requires access to the motion processor, include C-Motion.h. In addition, the required interfaces need to be defined as shown below. Only the required interfaces need to be included.

```
#define PMD_W32SERIAL_INTERFACE
// use this for a standard serial interface under Win9x/NT/2000/XP

#define PMD_PCI_INTERFACE
// use this for a standard PCI parallel interface under Win9x/NT/2000/XP
```

By customizing the base interface functions, C-Motion can be ported to virtually any hardware platform. An example would be a memory-mapped IO scheme that uses the parallel interface. This would be built using the PMDPar.c/.h source files as a basis.

The Magellan Motion Processor Developer's Kit board and the Prodigy-PCI Motion Card use the PCI interface chip provided by PLX Technology. To fully understand the interface mechanism, or to write your own interface software, you can download the PLX SDK. More information on the functionality and features can be found on the PLX website – http://www.plxtech.com – in the software development kits area.

C-Motion is a set of functions that encapsulate the motion processor command set. Every command has as its first parameter an "axis handle." The axis handle is a structure containing information about the interface to the motion processor and the axis number that the handle represents. Before communicating to the motion processor, the axis handle must be initialized using the following sequence of commands:

```
// the axis handles
PMDAxisHandle hAxis1, hAxis2;

// open interface to PMD processor and initialize handle to axis one
PMDSetupAxisInterface_PCI( &hAxis1, PMDAxis1, 0 );

// initialize handle to the second axis
PMDCopyAxisInterface( &hAxis2, &hAxis1, PMDAxis2 );
```

The above is an example of initializing communication using the parallel communication interface. Each interface .c source file contains an example of initializing the interface. Once the axis handle has been initialized, any of the motion processor commands can be executed.

The header file C-Motion.h includes the function prototypes for all motion processor commands as implemented in C-Motion. See this file for the required parameters for each command. For information about the operation and purpose of each command, see Chapter 4, *Instruction Reference*.

Many functions require additional parameters. Some standard values are defined by C-Motion and can be used with the appropriate functions. See PMDtypes.h for a complete list of defined types. An example of calling one of the C-Motion functions with the pre-defined types is shown below:

```
PMDSetBreakpoint(&hXAxis, PMDBreakpoint1, PMDAxis2, PMDBreakpointActionAbruptStop,
  PMDBreakpointActualPositionCrossed);
```

2.3.1 C-Motion Functions

The table below describes the functions that are provided by C-Motion in addition to the standard chip command set.

C-Motion functions	Arguments	Function description
PMDGetStatus	*axis_handle*	Returns the result of executing an IO status read. Only returns a valid result when the interface is PCI, ISA or Parallel.
PMDSetupAxisInterface_PCI	*axis_handle* *axis_number* *board_number*	Used to setup an axis interface connection for communicating over a PCI bus.
PMDSetupAxisInterface_ISA	*axis_handle* *axis_number* *board_number*	Used to setup an axis interface connection for communicating over an ISA (PC/104) bus.
PMDSetupAxisInterface_Serial	*axis_handle* *axis_number* *port_number*	Used to setup an axis interface connection for communicating over a RS232 or RS485 serial bus.
PMDSetupAxisInterface_CAN	*axis_handle* *axis_number* *board_number*	Used to setup an axis interface connection for communicating over a CAN bus.
PMDSetupAxisInterface_Parallel	*axis_handle* *axis_number* *board_address*	Low level function used to setup an axis interface for parallel communications in an embedded system.
PMDCloseAxisInterface	*axis_handle*	Should be called to terminate an interface connection.
PMDGetErrorMessage	*ErrorCode*	Returns a character string representation of the corresponding PMD chip or C-Motion error code.
GetCMotionVersion	*MajorVersion* *MinorVersion*	Returns the major and minor version number of C-Motion.
PMDHardReset	*axis_handle*	This function causes a “hard” reset of the motion processor. Unlike all other card-specific commands, this command is processed directly through the bus interface.
PMDReadDPRAM	*axis_handle* *data* *offset_in_dwords* *words_to_read*	This function reads directly from the onboard dual-port RAM via the bus interface (if applicable).
PMDWriteDPRAM	*axis_handle* *data* *offset_in_dwords* *words_to_write*	This function writes directly to the onboard dual-port RAM via the bus interface (if applicable).

2.4 Prodigy Motion Card Specific Functions

Several auxiliary functions are included in addition to the standard Magellan API commands for use with the Magellan-based Prodigy Motion Cards only. The functions are for configuring functions on the motion control board. The following table describes the functions. For more information, see the user's guide for your motion control card.

C-Motion function	Arguments	Function description
PMDMBWriteDigitalOutput	*axis_handle, write_value*	This function writes to the eight general-purpose digital I/O signals (digitalOut0-7). Write_value holds the eight signals in its low order 8 bits.
PMDMBReadDigitalInput	*axis_handle, read_value*	This function reads the value of the signals DigitalIn0-7, and returns them in the low order 8 bits of read_value.
PMDMBReadDigitalOutput	*axis_handle, read_value*	This function reads the value of the signals DigitalOut0-7, and returns them in the low order 8 bits of read_value.
PMDMBSetAmplifierEnable	*axis handle, mask, write_value*	This function writes to the 4 amplifier enable signals (AmpEnable1-4) using mask and write_value. When a 1 appears in mask, the corresponding bit position in write_value is written to the corresponding signal. The values for mask and write_value are all 0- shifted; that is, they are stored in the lowest order 4 bits.
PMDMBGetAmplifierEnable	*axis_handle, read_value*	This function reads the values of AmpEnable 1-4, and returns them in the low order 4 bits of read_value.
PMDMBSetDACOutputEnable	*axis handle, write_value*	This function sets the DACOutputEnable status. A written value of 1 enables DAC output, while a written value of 0 disables DAC output.
PMDMBGetDACOutputEnable	*axis_handle, read_value*	This function reads the value of the DACOutputEnable function. A value of 1 indicates DAC output enabled; a value of 0 indicates DAC output disabled.
PMDMBSetWatchDog	*axis handle*	This function writes to the correct value to the watchdog register, so that for the next 104 milliseconds the card will not be reset by the watchdog circuitry.
PMDMBGetResetCause	*axis_handle, reset_cause*	This function returns the reset cause in the variable reset_cause, reset_cause and also clears the reset condition.
PMDMBReadCardID	*axis_handle, card_ID*	This function returns the card ID, encoded as defined in the preceeding table.
PMDMBSetSSIRegister	*axis_handle resolution frequency*	Used to set up the DC-1000 parallel encoder option board for absolute SSI encoder support.
PMDMBGetSSIRegister	*axis_handle resolution frequency*	Returns the present setup of the DC-1000 SSI option board.
PMDMBGetSSIAbsolutePosition	*axis_handle position*	The function reads the DC-1000 SSI option board and returns the 32-bit absolute position reference.

3. VB-Motion

3.1 Introduction

VB-Motion provides a powerful Visual Basic object-oriented interface to the Magellan API and allows the developer to focus on writing high-level code to control the motion system. It can be easily integrated with any VB6 or VB.NET (including Microsoft VB.NET, Visual Studio.NET and Visual Studio 6) applications. The library supports communication to Magellan Developer's Kit Board and Magellan Motion Controller via serial (RS232/RS485) and CAN (IXXAT), and where applicable PCI, ISA and PC/104 parallel interfaces. There are two COM DLLs: PMDMP.dll and PMDUser.dll. Each of these DLLs contains a set of COM objects that provide access to the system. The following table describes the libraries.

COM Library	Description
PMDMP.dll	implements the PMDMPLib object which contains the communication objects for PCI and ISA interfaces (CommunicationPCI and CommunicationISA) and the motion processor command objects (MagellanObject, MagellanBoard and MagellanAxis).
PMDUser.dll	implements the PMDUserLib object which contains the communication objects for serial and IXXAT CAN interfaces (CommunicationSerial and CommunicationCAN).

VB-Motion includes the following features:

- Motion processor and Axis objects
- The ability to communicate to multiple PMD motion processors
- Supports PCI, ISA, serial, and CAN (IXXAT) interfaces

3.2 Files

The following table describes the example projects that are included with VB-Motion to provide a starting point for your custom motion software project.

Project	Description
AllCommands	Demonstrates the syntax for all available ION or Magellan commands
CANIO	Demonstrates how to setup a connection to the CAN interface
PCIIO	Demonstrates how to setup a connection to the PCI interface
SerialIO	Demonstrates how to setup a connection to the serial interface in point-to-point mode.

3.3 Using VB-Motion

In order to access the VB-Motion objects they must first be declared:

```
//Add this line when using the serial interface
Dim commSerial As PMDUserLib.CommunicationSerial

//Add this line when using the CAN interface
Dim commCAN As PMDUserLib.CommunicationCAN

//Add this line when using the PCI interface
Dim commPCI As PMDMPLib.CommunicationPCI

//Add this line when using the ISA interface
Dim commISA As PMDMPLib.CommunicationISA

//The standard motion processor objects
Dim magellanObj As PMDMPLib.MagellanObject
Dim boardObj As PMDMPLib.MagellanBoard
Dim axisObj As PMDMPLib.MagellanAxis
```

Before communicating to the motion processor, the communication object must be initialized using the following sequence of commands:

```
Set commSerial = New PMDUserLib.CommunicationSerial
commSerial.BaudRate = 57600

//Connect to COM1
commSerial.HostCOM = 1
....
```

The above is an example of initializing communication using the serial communication interface. Once the communication object has been initialized, create a Magellan object and a reference to one or more of the axes.

```
//Create an instance of the Magellan object
Set magellanObj = New MagellanObject

//Attach the serial driver
magellanObj.SetupCommunication commSerial

//Connect the events interface
magellanObj.EventListenerRegister Me

//Get axis 1
Set axis = magellanObj.Axes(PMD_AXIS_1)
```

Once the Magellan objects have been initialized, any of the motion processor commands can be executed. PMDMPLib contains all motion processor methods and properties. The property names are the same as listed in this manual but without the "Get" or "Set" prefix. The method names are the same as listed in this manual, but with the "Get" or "Set" prefix moved to a suffix (i.e., **GetBreakpoint** -> **BreakpointGet**). When a property is accessed the associated "Get" or "Set" command is sent to the processor to retrieve or send the data. Each property or method will throw an exception if an error occurs unless ModeSuppressExceptions is TRUE.

```
Dim valLong As Long

//To get a motion processor parameter use the following syntax
valLong = axis.Position

//To set a motion processor parameter use the following syntax
axis.Position = valLong
```

Some commands require additional parameters. Some standard values are defined by VB-Motion and can be used with the appropriate commands. Refer to PMDMPLib in the Object Viewer for a complete list of defined types. The following is an example of calling one of the VB-Motion methods with the pre-defined types.

```
Dim valBreakPoint As PMD_BREAKPOINT_ID
Dim valAxis As PMD_AXIS
Dim valBreakPointAction As PMD_BREAKPOINT_ACTION
Dim valBreakPointTrigger As PMD_BREAKPOINT_TRIGGER
valBreakPoint = PMD_BREAKPOINT_ID_1
valAxis = PMD_AXIS_1
valBreakPointAction = PMD_BREAKPOINT_ACTION_ABRUPT_STOP
valBreakPointTrigger = PMD_BREAKPOINT_TRIGGER_ACTUAL_POSITION_CROSSED
axis.BreakpointValue(valBreakPoint) = 1000
axis.BreakpointSet valBreakPoint, valAxis, valBreakPointAction, valBreakPointTrigger
```

The above example sets breakpoint 1 on axis 1 to trigger an abrupt stop if the actual position crosses 1000.

3.4 Prodigy Motion Card Specific Commands

Several auxiliary methods and properties are included, in addition to the standard Magellan API commands, for use with the Magellan-based Prodigy Motion Cards only. The commands are for configuring functions on the motion control board. The following table describes the commands. For more information, see the user's guide for your motion control card. These methods and properties are part of the Magellan Board object.

Name	Style	Description
ResetHardware	Method	This method causes a "hard" reset of the card. Unlike all other board-specific commands, this command is processed directly through the bus interface.
ReadDPRAM	Method	This method reads directly from the onboard dual-port RAM via the bus interface (if applicable).
WriteDPRAM	Method	This method writes directly to the onboard dual-port RAM via the bus interface (if applicable).
DigitalOutput	Prop r/w	Controls the eight general-purpose digital output signals (DigitalOut0-7).
DigitalInput	Prop Read	Contains the status of the eight general-purpose digital input signals (DigitalIn0-7).
CardID	Prop Read	Contains the card revision information.
DACOutputEnable	Prop r/w	Enables the DAC output hardware
ResetCause	Prop Read	Cause of the most recent reset.
AmplifierEnable	Prop r/w	Bit-mapped value of the four amplifier enable output signals (AmpEnable1-4)
WatchdogSet	Method	This method writes the correct value to the watchdog register, so that for the next 104 milliseconds the board will not be reset by the watchdog circuitry.

This page intentionally left blank.

4. Instruction Reference

4.1 How to Use This Reference

The instructions are arranged alphabetically, except that all "Set/Get" pairs (for example, **SetVelocity** and **GetVelocity**) are described together. Each description begins on a new page and most occupy no more than a single page. Each page is organized as follows:

Name	The instruction mnemonic is shown at the left, its hexadecimal code at the right.
Syntax	The instruction mnemonic (in bold) and its required arguments (in italic) are shown with all arguments separated by spaces.
Buffered	Certain parameters and other data written to the motion processor are buffered. That is, they are not acted upon until the next **Update** or **MultiUpdate** command is executed. These parameters are identified by the word "buffered" in the instruction heading.
Motor Types	The motor types to which this command applies. Supported motor types are printed in black; unsupported motor types for the command are greyed out.
Arguments	There are two types of arguments: encoded-field and numeric. Encoded-field arguments are packed into a single 16-bit data word, except for axis, which occupies bits 8–9 of the instruction word. The name of the argument (in italic) is that shown in the generic syntax. Instance (in italic) is the mnemonic used to represent the data value. Encoding is the value assigned to the field for that instance. For numeric arguments, the parameter value, the type (signed or unsigned integer), and the range of acceptable values are given. Numeric arguments may require one or two data words. For 32-bit arguments, the high-order part is transmitted first.
Packet Structure	This is a graphic representation of the 16-bit words transmitted in the packet: the instruction, which is identified by its name, followed by 1, 2, or 3 data words. Bit numbers are shown directly below each word. For each field in a word, only the high and low bits are shown. For 32-bit numeric data, the high-order bits are numbered from 16 to 31, the low-order bits from 0 to 15. The hex code of the instruction is shown in boldface. Argument names are shown in their respective words or fields. For data words, the direction of transfer—read or write—is shown at the left of the word's diagram. Unused bits are shaded. All unused bits must be 0 in data words and instructions sent (written) to the motion processor.
Description	Describes what the instruction does and any special information relating to the instruction.
Restrictions	Describes the circumstances in which the instruction is not valid, that is, when it should not be issued. For example, velocity, acceleration, deceleration, and jerk parameters may not be issued while an S-curve profile is being executed.
C-Motion API	The syntax of the C function call in the PMD C-Motion library that implements this motion processor command.
VB-Motion API	The Visual Basic syntax for the function in the PMD VB-Motion library that implements this motion processor command. Properties and methods are shown with their associated root object name separated by a period.
see	Refers to related instructions.

Syntax

AdjustActualPosition *axis position*

Motor Types

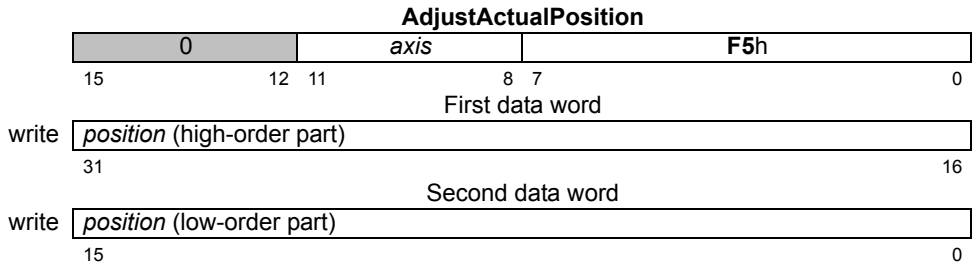
DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

	Type	Range	Scaling	Units
position	signed 32 bits	-2^{31} to $2^{31}-1$	unity	counts microsteps

Packet Structure



Description

The *position* specified as the parameter to **AdjustActualPosition** is summed with the actual position register (encoder position) for the specified *axis*. This has the effect of adding or subtracting an offset to the current actual position. At the same time, the commanded position is replaced by the new actual position value minus the position error. This prevents a servo "bump" when the new axis position is established. The destination position (see **SetPosition** (p. 133)) is also modified by this amount so that no trajectory motion will occur when a trajectory update is performed. In effect, this command establishes a new reference position from which subsequent positions can be calculated. It is commonly used to set a known reference position after a homing procedure.

On axes configured for stepping and microstepping motors, the position error is zeroed by this command.

AdjustActualPosition takes effect immediately; it is not buffered.

Restrictions

C-Motion API

```
PMDresult PMDAdjustActualPosition(PMDAxisInterface axis_intf,
                                  PMDint32 position)
```

VB-Motion API

```
MagellanAxis.AdjustActualPosition([in] position)
```

see

GetPositionError (p. 45), **GetActualVelocity** (p. 28), **Set/GetActualPositionUnits** (p. 70), **Set/GetActualPosition** (p. 68)

Syntax **ClearDriveFaultStatus** *axis*

Motor Types

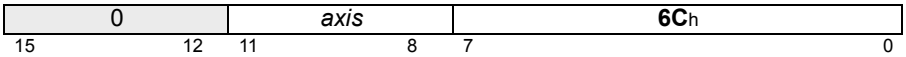
DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

Packet Structure

ClearDriveFaultStatus



Description **ClearDriveFaultStatus** clears all bits in the Drive Fault Status register. It should be executed after power-up, after using **GetDriveFaultStatus** to examine if any hard faults caused the power cycle.

Restrictions This command is not available in products that do not include drive amplifier support.

This command can only be executed when motor output is disabled (e.g., immediately after power-up or reset).

C-Motion API

```
PMDresult PMDClearDriveFaultStatus (PMDAxisInterface axis_intf)
```

VB-Motion API

```
MagellanAxis.ClearDriveFaultStatus()
```

see **GetDriveFaultStatus** (p. 37)

Syntax **ClearInterrupt** *axis*

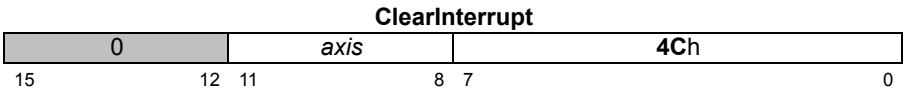
Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

Packet Structure



Description

ClearInterrupt resets the /HostInterrupt signal to its inactive state. If interrupts are still pending, the /HostInterrupt line will return to its active state within one chip cycle. See **Set/GetSampleTime** (p. 140) for information on chip cycle timing. This command is used after an interrupt has been recognized and processed by the host; it does not affect the Event Status register. The **ResetEventStatus** command should be issued prior to the **ClearInterrupt** command to clear the condition that generated the interrupt. The **ClearInterrupt** command has no effect if it is executed when no interrupts are pending.

When communicating using CAN, this command resets the interrupt message sent flag. When an interrupt is triggered on an ***axis***, a single interrupt message is sent and no further messages will be sent by that ***axis*** until this command is issued.

When serial or parallel communication is used, the axis number is not used.

Restrictions

For products without a /HostInterrupt line, this command is still applicable to the CAN communications. For products without a /HostInterrupt line or CAN communications, this command is not used.

C-Motion API

```
PMDresult PMDClearInterrupt (PMDAxisInterface axis_intf)
```

VB-Motion API

```
MagellanAxis.ClearInterrupt()
```

see **GetInterruptAxis** (p. 43), **Set/GetInterruptMask** (p. 113), **ResetEventStatus** (p. 64).

Syntax **ClearPositionError** *axis*

Motor Types

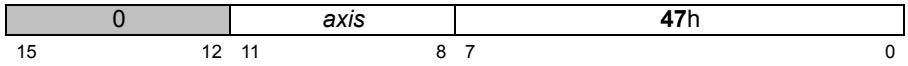
DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

Packet Structure

ClearPositionError



Description **ClearPositionError** sets the profile's commanded position equal to the actual position (encoder input), thereby clearing the position error for the specified ***axis***. This command can be used when the axis is at rest, or when it is moving.

Restrictions **ClearPositionError** is a buffered command. The new value set will not take effect until the next **Update** or **MultiUpdate** command, with the Trajectory bit set in the update mask commands.

This command should not be sent while the chip is executing a move using the S-curve profile mode.

C-Motion API

```
PMDresult PMDClearPositionError (PMDAxisInterface axis_intf)
```

VB-Motion API

```
MagellanAxis.ClearPositionError()
```

see **GetPositionError** (p. 45), **MultiUpdate** (p. 55), **Set/GetPositionErrorLimit** (p. 134), **Update** (p. 167)

Syntax **GetActiveMotorCommand** *axis*

Motor Types

DC Brush	Brushless DC	Microstepping	

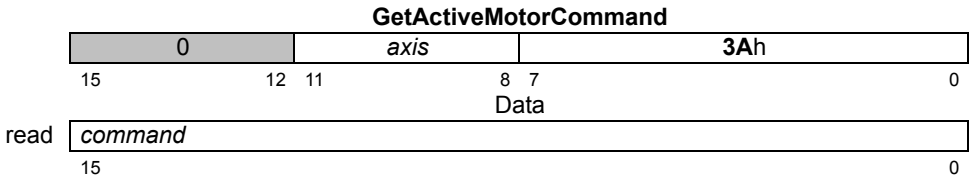
Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

Returned data

	Type	Range	Scaling	Units
command	signed 16 bits	-2^{15} *to* $2^{15}-1$	$100/2^{15}$	% output

Packet Structure



Description

GetActiveMotorCommand returns the value of the motor output command for the specified ***axis***. This is the input to the commutation or FOC current control. Its source depends on the motor type, as well as the operating mode of the ***axis***.

For brushless DC and DC brush motors: If position loop is enabled, it is the output of the position servo filter. If trajectory generator is enabled without the position loop, it is the output of the trajectory generator. If both trajectory generator and position loop are disabled, it is the contents of the motor output command register.

For microstepping motors: It is the contents of the motor output command register, subject to holding current reduction.

Restrictions

C-Motion API

```
PMDresult PMDGetActiveMotorCommand (PMDAxisInterface axis_intf,
                                    PMDint16* command)
```

VB-Motion API

```
Dim command as Short
command = MagellanAxis.ActiveMotorCommand
```

see **Set/GetMotorCommand** (p. 118), **Set/GetOperatingMode** (p. 122), **GetActiveOperatingMode** (p. 25)

Syntax **GetActiveOperatingMode** *axis*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

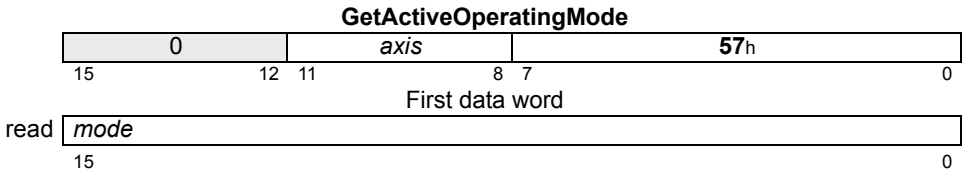
Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

Returned Data

	Type	
mode	unsigned 16 bits	bit field

Packet Structure



Description

GetActiveOperatingMode gets the actual operating mode that the ***axis*** is currently in. This may or may not be the same as the static operating mode, as safety responses or programmable conditions may change the **Active Operating Mode**. When this occurs, the **Active Operating Mode** can be changed to the programmed static operating mode using the **RestoreOperatingMode** command. The bit definitions of the operating mode are given below.

Name	Bit	Description
Axis Enabled	0	0: No *axis* processing, axis outputs in Reset state. 1: *axis* active.
Motor Output Enabled	1	0: *axis* motor outputs disabled. 1: *axis* motor outputs enabled.
Current Control Enabled	2	0: *axis* current control bypassed. 1: *axis* current control active.
—	3	Reserved
Position Loop Enabled	4	0: *axis* position loop bypassed. 1: *axis* position loop active.
Trajectory Enabled	5	0: trajectory generator disabled. 1: trajectory generator enabled.
—	6–15	Reserved

When the axis is disabled, no processing will be done on the axis, and the axis outputs will be at their reset states. When the axis motor output is disabled, the axis will function normally, but its motor outputs will be in their disabled state. When a loop is disabled (position or current loop), it operates by passing its input directly to its output, and clearing all internal state variables (such as integrator sums, etc.). When the trajectory generator is disabled, it operates by commanding zero (0) velocity.

Restrictions The possible modes of an axis are product specific, and in some cases axis specific. See the product user's guide for a description of what modes are supported on each axis.

C-Motion API

```
PMDresult PMDGetActiveOperatingMode(PMDAxisInterface axis_intf,
                                    PMDuint16* mode)
```

VB-Motion API

```
Dim mode as Short
mode = MagellanAxis.ActiveOperatingMode
```

see **GetOperatingMode** (p. 122), **RestoreOperatingMode** (p. 65), **Set/GetEventAction** (p. 102), **Set/GetBreakpoint** (p. 74)

Syntax **GetActivityStatus** *axis*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

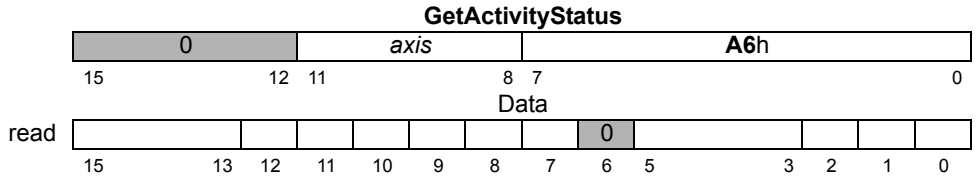
Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

Returned Data

	Type	
status	unsigned 16 bits	see below

Packet Structure



Description

GetActivityStatus reads the 16-bit Activity Status register for the specified ***axis***. Each of the bits in this register continuously indicate the state of the motion processor without any action on the part of the host. There is no direct way to set or clear the state of these bits, since they are controlled by the motion processor.

The following table shows the encoding of the data returned by this command.

Name	Bit(s)	Description
Phasing Initialized	0	Set to 1 if phasing is initialized (brushless DC axes only).
At Maximum Velocity	1	Set to 1 when the trajectory is at maximum velocity. This bit is determined by the trajectory generator, not the actual encoder velocity.
Tracking	2	Set to 1 when the axis is within the tracking window.
Current Profile Mode	3–5	Contains trajectory mode encoded as follows: bit 5 bit 4 bit 3 Profile Mode 0 0 0 Trapezoidal 0 0 1 Velocity Contouring 0 1 0 S-curve 0 1 1 Electronic Gear
—	6	Reserved; not used; may be 0 or 1.
Axis Settled	7	Set to 1 when the axis is settled.
Position Loop Enabled	8	Set to 1 when position loop or trajectory is enabled.
Position Capture	9	Set to 1 when a value has been captured by the high speed position capture hardware but has not yet been read.

Description (cont.)

Name	Bit(s)	Description
In-motion	10	Set to 1 when the trajectory generator is executing a profile.
In Positive Limit	11	Set to 1 when the positive limit switch is active.
In Negative Limit	12	Set to 1 when the negative limit switch is active.
Profile Segment	13–15	When the profile mode is S-curve, it contains the profile segment number 1–7 while profile is in motion, and contains a value of 0 when the profile is at rest. This field is undefined when using the Trapezoidal and Velocity Contouring profile modes.

Restrictions

C-Motion API

```
PMDresult PMDGetActivityStatus(PMDAxisInterface axis_intf,
                               PMDuint16* status)
```

VB-Motion API

```
Dim status as Short
status = MagellanAxis.ActivityStatus
```

see

GetEventStatus (p. 39), **GetSignalStatus** (p. 47), **GetDriveStatus** (p. 38)

Syntax **GetActualVelocity** *axis*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

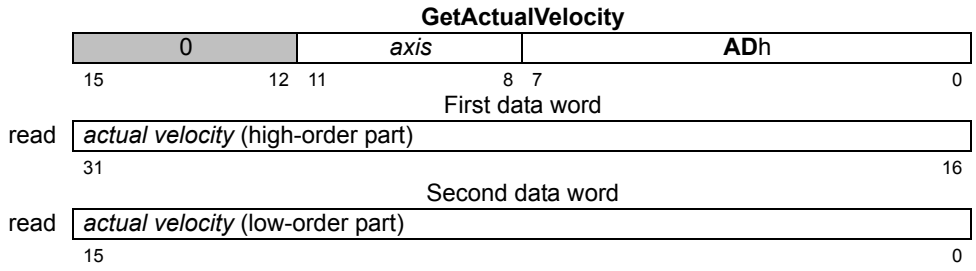
Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

Returned Data

	Type	Range	Scaling	Units
actual velocity	signed 32 bits	-2^{31} *to* $2^{31}-1$	$1/2^{16}$	counts/cycle

Packet Structure



Description

GetActualVelocity reads the value of the *actual velocity* for the specified *axis*. The *actual velocity* is derived by subtracting the actual position during the previous chip cycle from the actual position for this chip cycle. The result of this subtraction will always be integer because position is always integer. As a result the value returned by **GetActualVelocity** will always be a multiple of 65,536 since this represents a value of one in the 16.16 number format. The low word is always zero (0). This value is the result of the last encoder input, so it will be accurate to within one cycle.

Scaling example: If a value of 1,703,936 is retrieved by the **GetActualVelocity** command (high word: 01Ah, low word: 0h), this corresponds to a velocity of 1,703,936/65,536 or 26 counts/cycle.

Restrictions

C-Motion API

```
PMDresult PMDGetActualVelocity(PMDAxisInterface axis_intf,
                               PMDint32* velocity)
```

VBI-Motion API

```
Dim velocity as Long
velocity = MagellanAxis.ActualVelocity
```

see **GetCommandedVelocity** (p. 34), **GetActualPosition** (p. 68)

Syntax **GetBusVoltage** *axis*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

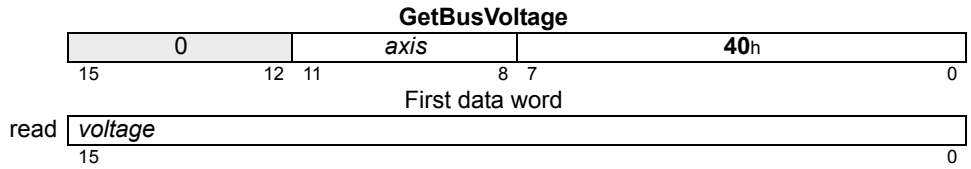
Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

Returned Data

	Type	Range	Scaling
voltage	unsigned 16 bits	0 *to* 2^16–1	1.3612 mv/count

Packet Structure



Description **GetBusVoltage** gets the most recent bus voltage reading from the ***axis***.

Restrictions **GetBusVoltage** is only available in products equipped with bus voltage sensors.

C-Motion API

```
PMDresult PMDGetBusVoltage(PMDAxisInterface axis_intf,
                           PMDuint16* voltage)
```

VB-Motion API

```
Dim voltage as Short
voltage = MagellanAxis.BusVoltage
```

see **Get/SetBusVoltageLimits** (p. 85)

Syntax **GetCaptureValue** *axis*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

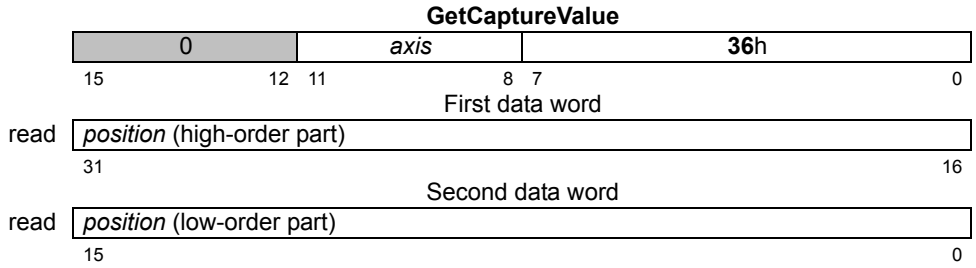
Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

Returned data

	Type	Range	Scaling	Units
position	signed 32 bits	-2^{31} *to* $2^{31}-1$	unity	counts microsteps

Packet Structure



Description **GetCaptureValue** returns the contents of the position capture register for the specified ***axis***. This command also resets bit 9 of the Activity Status register, thus allowing another capture to occur.

If actual position units is set to steps, the returned position will be in units of steps.

Restrictions

C-Motion API

```
PMDresult PMDGetCaptureValue(PMDAxisInterface axis_intf,
                             PMDint32* position)
```

VBI-Motion API

```
Dim position as Long
position = MagellanAxis.CaptureValue
```

see **Set/GetCaptureSource** (p. 88), **Set/GetActualPositionUnits** (p. 70), **GetActivityStatus** (p. 26)

Syntax GetChecksum

Motor Types

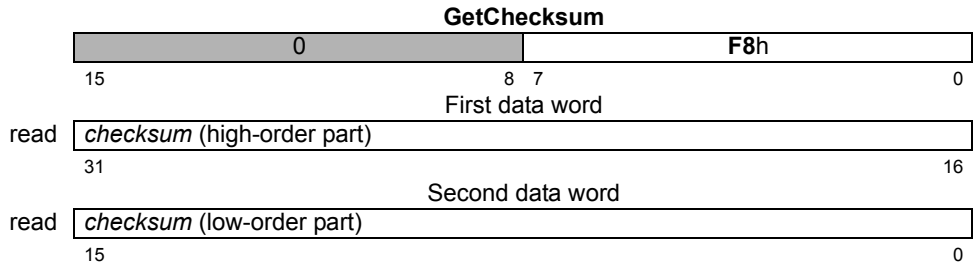
DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments None

Returned data

Name	Type
checksum	unsigned 32 bits

Packet Structure



Description **GetChecksum** reads the chips internal 32-bit *checksum* value. The return value is dependent on the silicon revision number of the motion processor.

Restrictions

C-Motion API

```
PMDresult PMDGetChecksum(PMDAxisInterface axis_intf,
                         PMDuint32* checksum)
```

VB-Motion API

```
Dim checksum as Long
checksum = MagellanObject.Checksum
```

see

Syntax **GetCommandedAcceleration** *axis*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

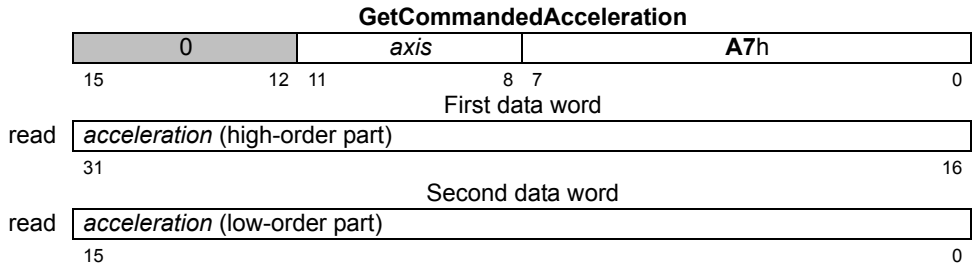
Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

Returned data

	Type	Range	Scaling	Units
acceleration	signed 32 bits	-2^{31} *to* $2^{31}-1$	$1/2^{16}$	counts/cycle^2 microsteps/cycle^2

Packet Structure



Description

GetCommandedAcceleration returns the commanded *acceleration* value for the specified *axis*. Commanded acceleration is the instantaneous acceleration value output by the trajectory generator.

Scaling example: If a value of 114,688 is retrieved using this command then this corresponds to 114,688/65,536 = 1.750 counts/cycle^2 acceleration value.

Restrictions

C-Motion API

```
PMDresult PMDGetCommandedAcceleration(PMDAxisInterface axis_intf,
                                      PMDint32* acceleration)
```

VB-Motion API

```
Dim acceleration as Long
acceleration = MagellanAxis.CommandedAcceleration
```

see **GetCommandedPosition** (p. 33), **GetCommandedVelocity** (p. 34)

Syntax **GetCommandedPosition** *axis*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

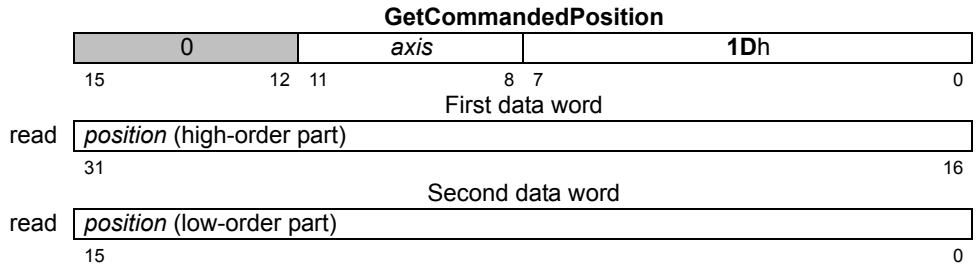
Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

Returned data

	Type	Range	Scaling	Units
position	signed 32 bits	-2^{31} *to* $2^{31}-1$	unity	counts microsteps

Packet Structure



Description **GetCommandedPosition** returns the commanded ***position*** for the specified ***axis***. Commanded position is the instantaneous position value output by the trajectory generator.

This command functions in all profile modes.

Restrictions

C-Motion API

```
PMDresult PMDGetCommandedPosition(PMDAxisInterface axis_intf,
                                  PMDint32* position)
```

VB-Motion API

```
Dim position as Long
position = MagellanAxis.CommandedPosition
```

see **GetCommandedAcceleration** (p. 32), **GetCommandedVelocity** (p. 34)

Syntax **GetCommandedVelocity** *axis*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

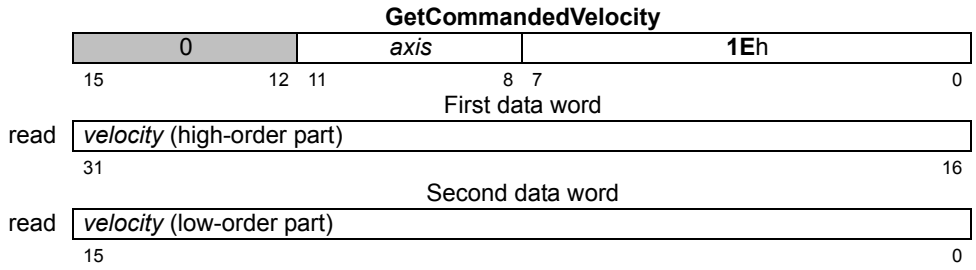
Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

Returned data

	Type	Range	Scaling	Units
velocity	signed 32 bits	-2^{31} *to* $2^{31}-1$	$1/2^{16}$	counts/cycle microsteps/cycle

Packet Structure



Description

GetCommandedVelocity returns the commanded ***velocity*** value for the specified ***axis***. Commanded velocity is the instantaneous velocity value output by the trajectory generator.

Scaling example: If a value of –1,234,567 is retrieved using this command (FFEDh in high word, 2979h in low word) then this corresponds to –1,234,567/65,536 = –18.8380 counts/cycle velocity value.

Restrictions

C-Motion API

```
PMDresult PMDGetCommandedVelocity(PMDAxisInterface axis_intf,
                                  PMDint32* velocity)
```

VB-Motion API

```
Dim velocity as Long
velocity = MagellanAxis.CommandedVelocity
```

see **GetCommandedAcceleration** (p. 32), **GetCommandedPosition** (p. 33)

Syntax **GetCurrentLoopValue** *axis loopnum node*

Motor Types

DC Brush	Brushless DC	Microstepping	

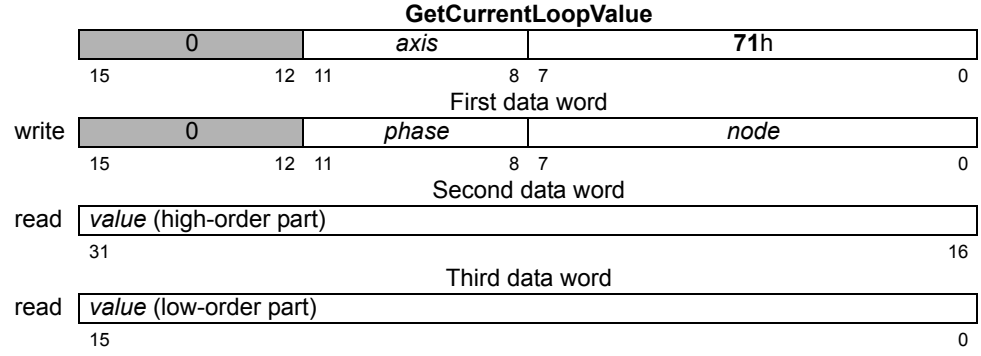
Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
phase	*Phase A*	0
	Phase B	1
node	*Reference*	0
	Actual Current	1
	Error	2
	Integrator Sum	3
	— (Reserved)	4
	Integral Contribution	5
	Output	6

Returned data

	Type	Range/Scaling
value	signed 32 bits	see below

Packet Structure



Description

GetCurrentLoopValue is used to read the value of a ***node*** in one of the digital current loops. See the product user's guide for more information on the location of each ***node*** in the current loop processing. Though the data returned is signed 32 bits regardless of the ***node***, the range and format vary depending on the ***node***, as follows:

Node	Range	Scaling	Units
Reference	–2^15 to 2^15–1	100/2^14	% max current
Actual Current	–2^15 to 2^15–1	100/2^14	% max current
Error	–2^15 to 2^15–1	100/2^14	% max current
Integrator Sum	–2^31 to 2^31–1	100/2^14	(% max current)* current loop cycles
Integral Contribution	–2^31 to 2^31–1	100/2^14	% max current
Output	–2^15 to 2^15–1	100/2^14	% max current

Description (cont.)

All of the ***nodes*** have units of % maximum current, and most have scaling of 100/2^14. That is, a value of 2^14 corresponds to 100% maximum current. The range is extended to allow for overshoot in excess of maximum peak current, and thus values can be more than 100% of the maximum output current.

The Integrator Sum is a signed 32-bit number, with scaling of 100/2^14. That is, a current error of 100% maximum, present for 16 current loop cycles, will result in an integrator sum of 16*(100%)*2^14/100 = 2^18. Current loop cyles are not the same as position loop servo cycles. The current loop runs at 20 kHz, regardless of the servo cycle time.

Restrictions

This command is only supported in products that include digital current control, and when the current control mode is Phase A /B.

C-Motion API

```
PMDresult PMDGetCurrentLoopValue(PMDAxisInterface axis_intf,
                                 PMDuint8 phase,
                                 PMDuint8 node,
                                 PMDint32* value)
```

VB-Motion API

```
MagellanAxis.CurrentLoopValue ([in] phase,
                               [in] node,
                               [out] value
```

see

Set/GetCurrentLoop (p. 93), **Set/GetCurrentControlMode** (p. 90)

Syntax **GetDriveFaultStatus** *axis*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

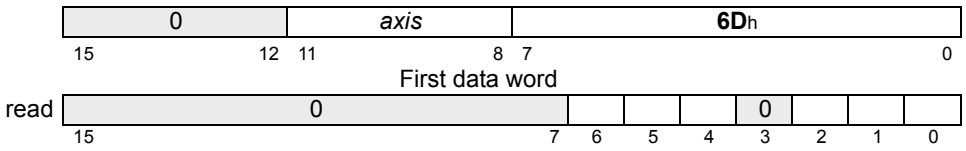
Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

Returned Data

	Type	
status	unsigned 16 bits	see below

Packet Structure

GetDriveFaultStatus



Description **GetDriveFaultStatus** gets the Drive Fault Status register, which contains a bitmap showing all hard faults that have occurred since the Drive Fault Status register was last cleared. This register is kept in non-volatile memory, so that a record of hard faults is retained even through power cycles, which will be done upon any hard fault event.

The table below shows the bit definitions of the Drive Fault Status register.

Name	Bit
Overcurrent Fault	0
Ground Fault	1
External Logic Fault	2
— (Reserved)	3
Internal Logic Fault	4
Overvoltage Fault	5
Undervoltage Fault	6
— (Reserved)	7–15

Events 0 through 4 are hard faults. If one of these occur, the unit will shut down, and power must be cycled. Upon power-up, **GetDriveFaultStatus** should be used to check which, if any, hard fault may have caused the previous power cycle. After querying the Drive Fault Status register, it should be cleared using **ClearDriveFaultStatus**. If this is not done, the bits will be retained in non-volatile memory, which will diminish the ability to detect the cause of any subsequent hard faults.

Events 5 and 6 will not cause the system to shut down. Instead, they will cause the system to change to the disabled state, and will cause the Bus Voltage Fault bit in **GetEventStatus** to be set. Normally, the Drive Fault Status register does not need to be monitored. In the case of Bus Voltage Fault in **GetEventStatus**, however, the Drive Fault Status register can be used to distinguish the error between overvoltage and undervoltage. The Overvoltage Fault and Undervoltage Fault bits are cleared upon power-up.

Restrictions This command is not available in products without drive amplifier support.

C-Motion API

```
PMDresult PMDGetDriveFaultStatus(PMDAxisInterface axis_intf,
                                 PMDuint16* status)
```

VB-Motion API

```
Dim status as Short
status = MagellanAxis.DriveFaultStatus
```

see **ClearDriveFaultStatus** (p. 21)

Syntax **GetDriveStatus** *axis*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

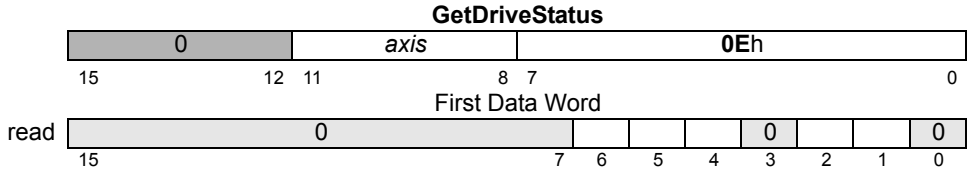
Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

Returned data

	Type	
status	unsigned 16 bits	see below

Packet Structure



Description

GetDriveStatus reads the Drive Status register for the specified ***axis***. All of the bits in this status word are set and cleared by the motion processor. They are not settable or clearable by the host. The bits represent states or conditions in the motion processor that are of a transient nature.

Name	Bit(s)	Description
—	0	Reserved; not used; may be 0 or 1.
In Foldback	1	Set to 1 when the unit is in the current foldback state–the output current is limited by the foldback limit.
Overtemperature	2	Set to 1 when the overtemperature condition is present.
—	3	Reserved; not used; may be 0 or 1.
In Holding	4	Set to 1 when the unit is in the holding current state–the output current is limited by the holding current limit.
Overvoltage	5	Set to 1 when the overvoltage condition is present.
Undervoltage	6	Set to 1 when the undervoltage condition is present.
—	7–15	Reserved; not used; may be 0 or 1.

Restrictions The bits available in this register depend upon the products. See the product user's guide.

C-Motion API

```
PMDresult PMDGetDriveStatus(PMDAxisInterface axis_intf,
                            PMDuint16* status)
```

VB-Motion API

```
Dim status as Short
status = MagellanAxis.DriveStatus
```

see

Syntax **GetEventStatus** *axis*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

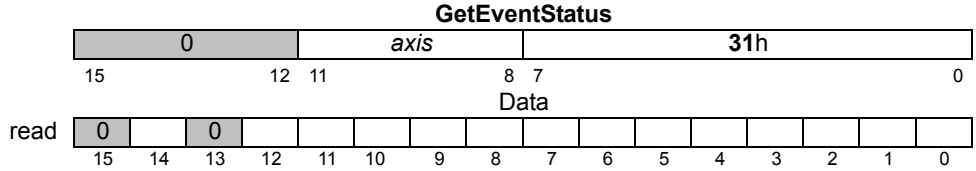
Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

Returned data

	Type	
status	unsigned 16 bits	see below

Packet Structure



Description **GetEventStatus** reads the Event Status register for the specified ***axis***. All of the bits in this status word are set by the motion processor and cleared by the host. To clear these bits, use the **ResetEventStatus** command. The following table shows the encoding of the data returned by this command.

Name	Bit(s)	Description
Motion Complete	0	Set to 1 when motion has completed. **SetMotionCompleteMode** determines if this bit is based on the trajectory generator position or the encoder position.
Wrap-around	1	Set to 1 when the actual (encoder) position has wrapped from maximum allowed position to minimum, or vice versa.
Breakpoint 1	2	Set to 1 when breakpoint 1 has been triggered.
Capture Received	3	Set to 1 when a position capture has occurred.
Motion Error	4	Set to 1 when a motion error has occurred.
Positive Limit	5	Set to 1 when the axis has entered a positive limit switch.
Negative Limit	6	Set to 1 when the axis has entered a negative limit switch.
Instruction Error	7	Set to 1 when an instruction error has occurred.
Disable	8	Set to 1 when "disable" due to user /Enable line has occurred.
Overtemperature Fault	9	Set to 1 when overtemperature condition has occurred.
Bus Voltage Fault	10	Set to 1 when an overvoltage or undervoltage condition has occurred.
Commutation error	11	Set to 1 when a commutation error has occurred.
Current Foldback	12	Set to 1 when current foldback has occurred.
—	13	Reserved; not used; may be 0 or 1.
Breakpoint 2	14	Set to 1 when breakpoint 2 has been triggered.
—	15	Reserved; not used; may be 0 or 1.

Restrictions Bits 8, 9, 10, and 12 are not implemented in products that do not include drive amplifier support. In this case, they are reserved—may be 0 or 1.

C-Motion API

```
PMDresult PMDGetEventStatus(PMDAxisInterface axis_intf,
                            PMDuint16* status)
```

VB-Motion API

```
Dim status as Short
status = MagellanAxis.EventStatus
```

see **GetActivityStatus** (p. 26), **GetSignalStatus** (p. 47), **GetDriveStatus** (p. 38)

Syntax **GetFOCValue** *axis loop node*

Motor Types

	Brushless DC	Microstepping	

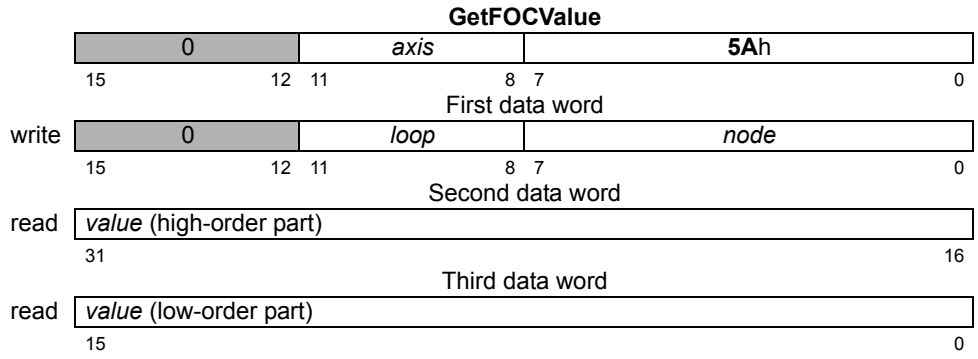
Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
loop	*Direct (D)*	0
	Quadrature (Q)	1
node	*Reference (D,Q)*	0
	Feedback (D,Q)	1
	Error (D,Q)	2
	Integrator Sum (D,Q)	3
	— (Reserved)	4
	Integral Contribution (D,Q)	5
	Output (D,Q)	6
	FOC Output (Alpha,Beta)	7
	Actual Current (A,B)	8

Returned data

	Type	Range/Scaling
value	signed 32 bits	see below

Packet Structure



Description

GetFOCValue is used to read the value of a ***node*** of the FOC current control. See the product user's guide for more information on the location of each ***node*** in the FOC current control algorithm. Though the data returned is signed 32 bits regardless of the ***node***, the range and format vary depending on the ***node***, as follows:

Node	Range	Scaling	Units
Reference (D,Q)	–2^15 to 2^15–1	100/2^14	% max current
Feedback (D,Q)	–2^18 to 2^18–1	100/2^14	% max current
Error (D,Q)	–2^15 to 2^15–1	100/2^14	% max current
Integrator Sum (D,Q)	–2^31 to 2^31–1	100/2^14	(% max current)* current loop cycles
Integral Contribution (D,Q)	–2^31 to 2^31–1	100/2^14	% max current
Output (D,Q)	–2^15 to 2^15–1	100/2^14	% PWM
FOC Output (Alpha,Beta)	–2^15 to 2^15–1	100/2^14	% PWM
Actual Current (A,B)	–2^15 to 2^15–1	100/2^14	% max current

Description (cont.)

Most of the ***nodes*** have units of % maximum current, and most have a scaling of 100/2^14. That is, a value of 2^14 corresponds to 100% maximum current. The range is extended to allow for overshoot in excess of maximum peak current, and thus values can be more than 100% of the maximum output current.

The ***Integrator Sum*** is a signed 32-bit number, with scaling of 100/2^14. That is, a current of 100% maximum, present for 16 current loop cycles, will result in an integrator sum of 16*(100%)*2^14/100 = 2^18. Current loop cycles are not the same as position loop servo cycles. The current loop runs at 20 kHz, regardless of the servo cycle time.

Restrictions

This command is only supported in products that include digital current control, and when the current control mode is set to FOC.

C-Motion API

```
PMDresult PMDGetFOCValue (PMDAxisInterface axis_intf,
                          PMDuint8 loop,
                          PMDuint8 node,
                          PMDint32* value)
```

VB-Motion API

```
MagellanAxis.FOCValue ( [in] loop,
                        [in] node,
                        [out] value )
```

see

Set/GetFOC (p. 106), **Set/GetCurrentControlMode** (p. 90)

Syntax **GetInstructionError**

Motor Types

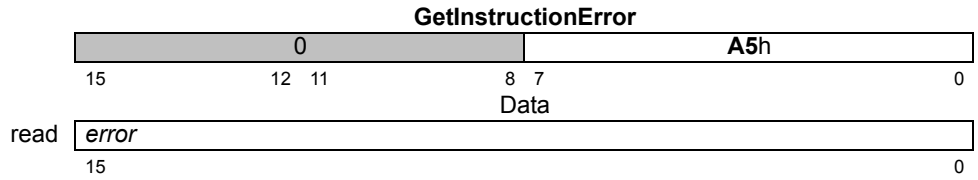
DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments None

Returned data

	Type	Range
error	unsigned 16 bits	0 *to* 11h

Packet Structure



Description **GetInstructionError** returns the code for the last instruction error, and then resets the error to zero (0). Generally, this command is issued only after the instruction error bit in the Event Status register indicates there was an instruction error. It also resets the Instruction error bit in the I/O status read word to zero (0).

The error codes are encoded as defined below:

Error Code	Encoding
No error	0
Processor reset	1
Invalid instruction	2
Invalid axis	3
Invalid parameter	4
Trace running	5
— (Reserved)	6
Block out of bounds	7
Trace buffer zero (0)	8
Bad serial checksum	9
— (Reserved)	Ah
Invalid negative value	Bh
Invalid parameter change	Ch
Invalid move after event-triggered stop	Dh
Invalid move into limit	Eh
Invalid Operating Mode restore after event-triggered change	10h
Invalid Operating Mode for command	11h

Restrictions

C-Motion API

```
PMDresult PMDGetInstructionError (PMDAxisInterface axis_intf,
                                  PMDuint16* error)
```

VB-Motion API

```
Dim error as Short
error = MagellanObject.InstructionError
```

see **GetEventStatus** (p. 39), **ResetEventStatus** (p. 64)

Syntax GetInterruptAxis

Motor Types

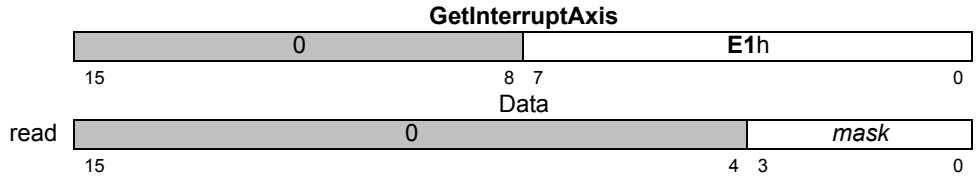
DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments None

Returned data

Name	Instance	Encoding
mask	*None*	0
	Axis1 Mask	1
	Axis2 Mask	2
	Axis3 Mask	4
	Axis4 Mask	8

Packet Structure



Description **GetInterruptAxis** returns a field that identifies all axes with pending interrupts. Axis numbers are assigned to the low-order four bits of the returned word, with bits corresponding to interrupting axes set to 1. If there are no pending interrupts, the returned word is zero (0). If any axis has a pending interrupt, the /HostInterrupt signal will be in an active state.

Restrictions This command is only useful for products with /HostInterrupt pin. When using CAN events for interrupt event notification, the interrupting axis is sent as part of the CAN event.

C-Motion API

```
PMDresult PMDGetInterruptAxis(PMDAxisInterface axis_intf,
                              PMDuint16* mask)
```

VB-Motion API

```
Dim mask as Short
mask = MagellanObject.InterruptAxis
```

see **ClearInterrupt** (p. 22), **Set/GetInterruptMask** (p. 113)

Syntax **GetPhaseCommand** *axis phase*

Motor Types

	Brushless DC	Microstepping	

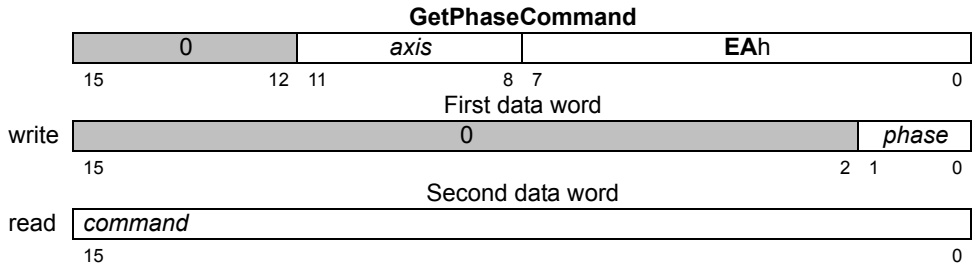
Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
phase	*Phase A*	0
	Phase B	1
	Phase C	2

Returned data

	Type	Range	Scaling	Units
command	signed 16 bits	-2^{15} *to* $2^{15}-1$	$100/2^{15}$	% output

Packet Structure



Description **GetPhaseCommand** returns the value of the commutated phase command for phase A, B, or C of the specified *axis*. These are the phase command values directly output to the current loop or motor after commutation.

Scaling example: If a value of –4,489 is retrieved (EE77h) for a given axis and phase, then this corresponds to –4,489*100/32,767 = –13.7% of full-scale output.

Restrictions *Phase C* is only valid when the motor type has been set for a 3-phase commutation.

This command has no meaning when current control mode is set to FOC whether or not the current loops are enabled.

When the current control mode is set to *Phase A /B* current loops, the values are the inputs to the current loops. When current loops are disabled, the value is the motor output command.

C-Motion API

```
PMDresult PMDGetPhaseCommand(PMDAxisInterface axis_intf,
                             PMDuint16 phase,
                             PMDint16* command)
```

VB-Motion API

```
Dim command as Short
command = MagellanAxis.PhaseCommand( phase )
```

see **SetCurrentControlMode** (p. 90)

Syntax **GetPositionError** *axis*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

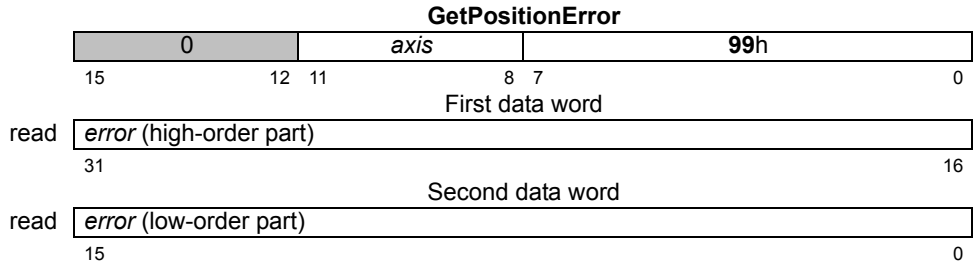
Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

Returned data

	Type	Range	Scaling	Units
error	signed 32 bits	-2^{31} *to* $2^{31}-1$	unity	counts microsteps

Packet Structure



Description **GetPositionError** returns the position error of the specified ***axis***. The error is the difference between the actual position (encoder position) and the commanded position (instantaneous output of the trajectory generator). When used with the motor type set to microstepping or pulse & direction, the error is defined as the difference between the encoder position (represented in microsteps or steps) and the commanded position (instantaneous output of the trajectory generator).

Restrictions

C-Motion API

```
PMDresult PMDGetPositionError(PMDAxisInterface axis_intf,
                              PMDint32* error)
```

VB-Motion API

```
Dim error as Long
error = MagellanAxis.PositionError
```

see **Set/GetPosition** (p. 133), **Set/GetPositionErrorLimit** (p. 134)

Syntax **GetPositionLoopValue** *axis node*

Motor Types

DC Brush	Brushless DC		

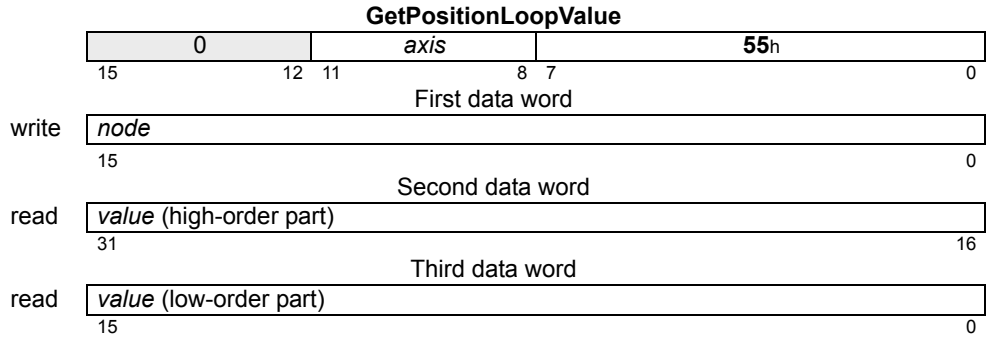
Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
node	*Integrator Sum*	0
	Integral Contribution	1
	Derivative	2
	Biquad1 Input	3
	Biquad2 Input	4

Returned data

	Type	Range/Scaling
value	signed 32 bits	see below

Packet Structure



Description

GetPositionLoopValue is used to find the value of a ***node*** in the position loop. See the product user's guide for more information on the location of each ***node*** in the position loop processing. Though the data returned is signed 32 bits regardless of the ***node***, the range and format varies depending on the ***node***, as follows:

Node	Range	Scaling	Units
Integrator Sum	–2^31 to 2^31–1	unity	(counts or microsteps)*cycles
Integral Contribution	–2^31 to 2^31–1	100*Kout/(2^16)	% Output
Derivative	–2^15 to 2^15–1	unity	(counts or microsteps)/cycles
Biquad1 Input	–2^15 to 2^15–1	unity	counts or microsteps
Biquad2 Input	–2^15 to 2^15–1	unity	counts or microsteps

Restrictions

C-Motion API

```
PMDresult PMDGetPositionLoopValue (PMDAxisInterface axis_intf,
                                   PMDuint16 node,
                                   PMDint32* value)
```

VB-Motion API

```
Dim value as Long
value = MagellanAxis.PositionLoopValue( node )
```

see **Set/GetPositionLoop** (p. 135)

Syntax **GetSignalStatus** *axis*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

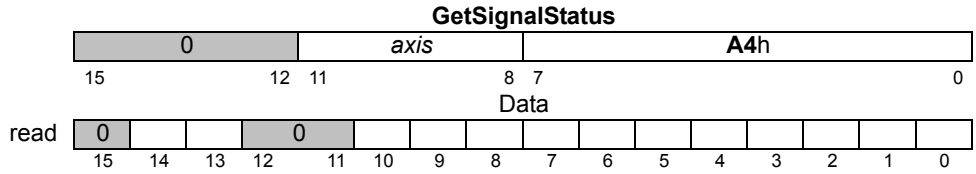
Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

Returned data

	Type
see below	unsigned 16 bits

Packet Structure



Description

GetSignalStatus returns the contents of the Signal Status register for the specified ***axis***. The Signal Status register contains the value of the various hardware signals connected to each axis of the motion processor. The value read is combined with the Signal Sense register (see **SetSignalSense** (p. 146)) and then returned to the user. For each bit in the Signal Sense register that is set to 1, the corresponding bit in the **GetSignalStatus** command will be inverted. Therefore, a low signal will be read as 1, and a high signal will be read as a 0. Conversely, for each bit in the Signal Sense register that is set to 0, the corresponding bit in the **GetSignalStatus** command is not inverted. Therefore, a low signal will be read as 0, and a high signal will be read as a 1.

All of the bits in the **GetSignalStatus** command are inputs, except for AxisOut and FaultOut. The value read for these bits is equal to the value output by the AxisOut and FaultOut mechanisms. See **SetAxisOutMask** (p. 72) and **SetFaultMask** (p. 104) for more information. The bit definitions are as follows:

Description	Bit Number
Encoder A	0
Encoder B	1
Encoder Index	2
Capture Input	3
Positive Limit	4
Negative Limit	5
AxisIn	6
Hall A	7

Description	Bit Number
Hall B	8
Hall C	9
AxisOut	10
— (Reserved)	11–12
/Enable In	13
FaultOut	14
— (Reserved)	15

Restrictions

Depending on the product, some signals may not be present. See the product user's guide. In ION products, when the capture source is set to Index, the Encoder Index input will be present as both the Encoder Index and the Capture Input bits.

C-Motion API

```
PMDresult PMDGetSignalStatus(PMDAxisInterface axis_intf,
                             PMDuint16* status)
```

VB-Motion API

```
Dim status as Short
status = MagellanAxis.SignalStatus
```

see **GetActivityStatus** (p. 26), **GetEventStatus** (p. 39), **GetSignalSense** (p. 146)

Syntax **GetTemperature** *axis*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

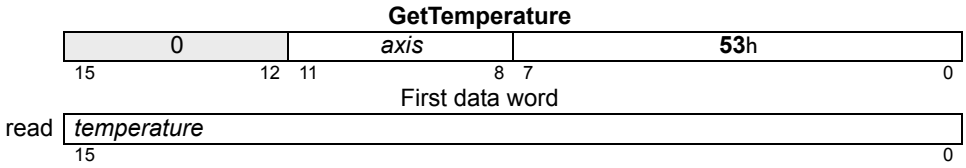
Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

Returned Data

	Type	Range	Scaling	Units
temperature	signed 16 bits	–2^15 *to* 2^15–1	2^8	°C

Packet Structure



Description **GetTemperature** gets the most recent temperature reading from the temperature sensor(s) monitoring the ***axis***.

Restrictions **GetTemperature** is only available in products equipped with temperature sensors. If ***axis*** has more than one temperature sensor, the temperature returned will be the average value of all sensor readings.

C-Motion API

```
PMDresult PMDGetTemperature(PMDAxisInterface axis_intf,
                            PMDint16* temperature)
```

VB-Motion API

```
Dim temperature as Short
temperature = MagellanAxis.Temperature
```

see **Get/SetOvertemperatureLimit** (p. 125)

Syntax **GetTime**

Motor Types

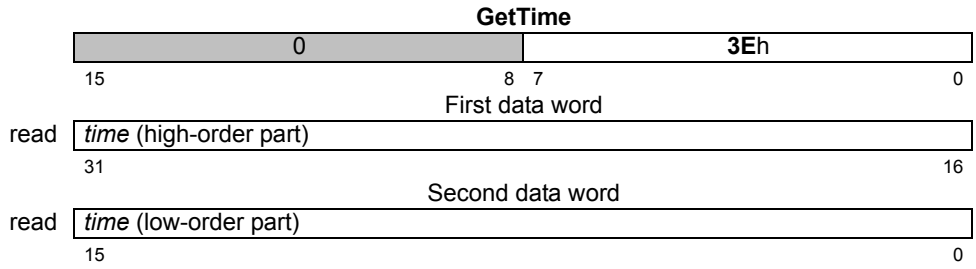
DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments None

Returned data

Name	Type	Range	Scaling	Units
time	unsigned 32 bits	0 *to* $2^{32}-1$	unity	cycles

Packet Structure



Description **GetTime** returns the number of cycles which have occurred since the motion processor was last reset. The time per cycle is determined by **SetSampleTime**.

Restrictions Time stops advancing when no axes are enabled.

C-Motion API

```
PMDresult PMDGetTime(PMDAxisInterface axis_intf,
                     PMDuint32* time)
```

VB-Motion API

```
Dim time as Long
time = MagellanObject.Time
```

see **Set/GetSampleTime** (p. 140)

Syntax GetTraceCount

Motor Types

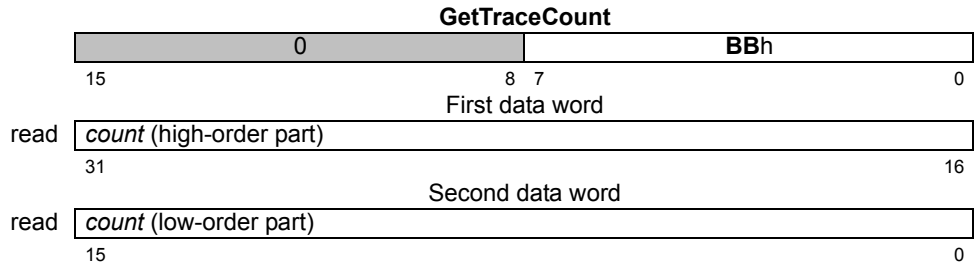
DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments None

Returned data

Name	Type	Range	Scaling	Units
count	unsigned 32 bits	0 *to* $2^{32}-1$	unity	samples

Packet Structure



Description **GetTraceCount** returns the number of points (variable values) stored in the trace buffer since the beginning of the trace.

Restrictions If the trace mode is set to "rolling" and the buffer wraps, **GetTraceCount** returns the number of samples in the filled buffer.

C-Motion API

```
PMDresult PMDGetTraceCount(PMDAxisInterface axis_intf,
                           PMDuint32* count)
```

VB-Motion API

```
Dim count as Long
count = MagellanObject.TraceCount
```

see **ReadBuffer** (p. 58), **Set/GetTraceStart** (p. 155), **Set/GetTraceStop** (p. 157), **Set/GetBufferLength** (p. 81)

Syntax GetTraceStatus

Motor Types

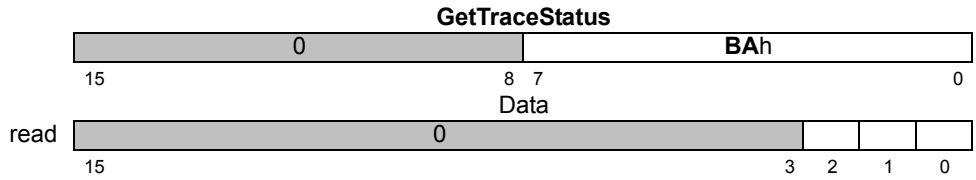
DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments None

Returned data

Name	Type
see below	unsigned 16 bits

Packet Structure



Description **GetTraceStatus** returns the trace status. The definitions of the individual status bits are as follows:

Name	Bit Number	Description
Mode	0	Set to 0 when trace is in one-time mode, 1 when in rolling mode.
Activity	1	Set to 1 when trace is active (currently tracing), 0 if trace not active.
Data Wrap	2	Set to 1 when trace has wrapped, 0 if it has not wrapped. If 0, the buffer has not yet been filled, and all recorded data is intact. If 1, the trace has wrapped to the beginning of the buffer; any previous data may have been overwritten if not explicitly retrieved by the host using the **ReadBuffer** command while the trace is active.

Restrictions

C-Motion API

```
PMDresult PMDGetTraceStatus(PMDAxisInterface axis_intf,
                            PMDuint16* status)
```

VB-Motion API

```
Dim status as Short
status = MagellanObject.TraceStatus
```

see **Set/GetTraceStart** (p. 155), **Set/GetTraceMode** (p. 153)

Syntax **GetVersion**

Motor Types

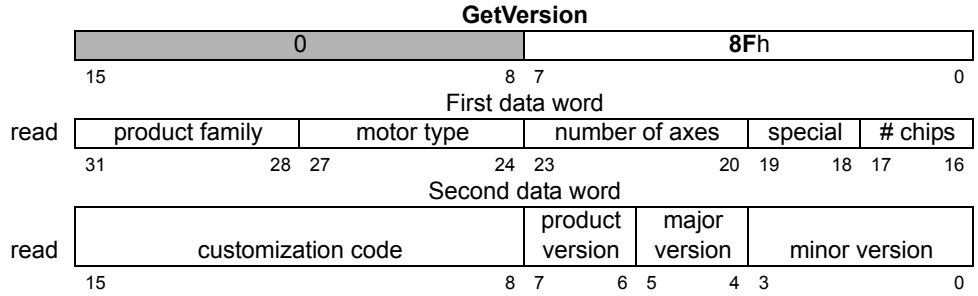
DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments None

Returned data

Name	Type
version	unsigned 32 bits

Packet Structure



Description **GetVersion** returns product information encoded as shown in the preceding packet structure diagram. Individual data fields are encoded as defined in the following table.

Name	Description	Encoding
product family	Navigator	2
	Pilot	3
	Magellan	5
	ION	9
motor type	Servo	1
	Brushless	3
	Microstepping	4
	Pulse & Direction	5
	All Motor Types	8
	ION–Any Motor Type (read only)	9
number of axes	Maximum number of supported axes	1 to 15
special	(Reserved)	0 to 3
# chips		0 to 3
customization code	None	0
	Other	1 to 255
product version		0 to 3
major s/w version		0 to 3
minor s/w version		0 to 15

Restrictions Note that in the C-Motion function **PMDGetVersion**, the special attributes value and the chip count values are combined and returned in a single parameter (`special_and_chip_count`). Chip count is encoded in bits 0–1 of this value; special is encoded in bits 2–3. Likewise for the `major` parameter. The `major` version is encoded in bits 0-1 and the product version is encoded in bits 2-3.

C-Motion API

```
PMDresult PMDGetVersion(PMDAxisInterface axis_intf,
                        PMDuint16* family,
                        PMDuint16* motorType,
                        PMDuint16* numberAxes,
                        PMDuint16* special_and_chip_count,
                        PMDuint16* custom,
                        PMDuint16* major,
                        PMDuint16* minor)
```

VB-Motion API

```
Dim version as Long
version  = MagellanObject.Version
```

see

Syntax **InitializePhase** *axis*

Motor Types

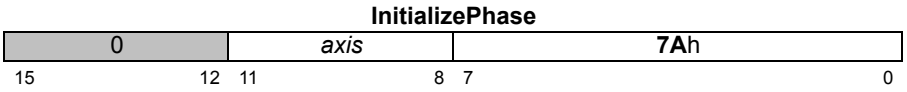
	Brushless DC		

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

Returned data None

Packet Structure



Description **InitializePhase** initializes the phase angle for the specified ***axis*** using the mode (Hall-based or algorithmic) specified by the **SetPhaseInitializationMode** command.

Restrictions **Warning: If the phase initialization mode has been set to algorithmic, then, after this command is sent, the motor may suddenly move in an uncontrolled manner.**

C-Motion API `PMDresult` **`PMDInitializePhase`**`(PMDAxisInterface` *`axis_intf`*`)`

VB-Motion API **`MagellanAxis.InitializePhase`**`()`

see **GetPhaseCommand** (p. 44), **Set/GetCommutationMode** (p. 89)

Syntax **MultiUpdate** *mask*

Motor Types

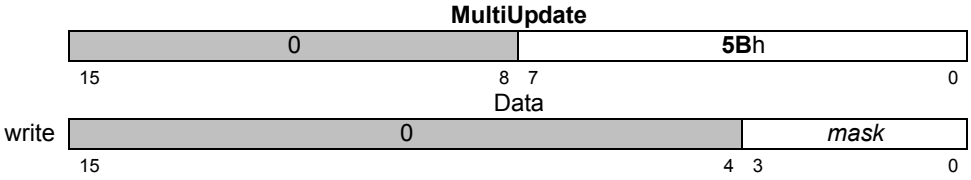
DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding
mask	*None*	0
	Axis1 Mask	1
	Axis2 Mask	2
	Axis3 Mask	4
	Axis4 Mask	8

Returned data None

Packet Structure



Description **MultiUpdate** causes an update to occur on all axes whose corresponding bit is set to 1 in the ***mask*** argument. After this command is executed, all axes which are selected using the mask will perform an **Update**. The paramater groups that are copied from their buffered versions into the corresponding run-time registers is determined by the update mask of each ***axis***, as shown in the table below.

Group	Command/Parameter
Trajectory	Acceleration
	Deceleration
	Gear Ratio
	Jerk
	Position
	Profile Mode
	Stop Mode
	Velocity
	ClearPositionError
Position Servo	Derivative Time
	Integration Limit
	Kaff
	Kd
	Ki
	Kp
	Kvff
	Kout
	Motor Command
Current Loops	Integration Limit
	Ki
	Kp

Restrictions

C-Motion API

```
PMDresult PMDMultiUpdate(PMDAxisInterface axis_intf,
                         PMDuint16 mask)
```

VB-Motion API

```
MagellanObject.MultiUpdate( [in] mask )
```

see **Update** (p. 167), **Set/GetUpdateMask** (p. 163)

NoOperation 00h

Syntax **NoOperation**

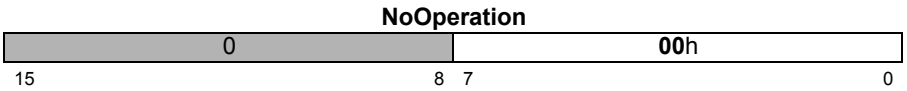
Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments None

Returned data None

Packet Structure



Description The **NoOperation** command has no effect on the motion processor.

Restrictions

C-Motion API `PMDresult` **`PMDNoOperation`**`(PMDAxisInterface` *`axis_intf`*`)`

VB-Motion API **`MagellanObject.NoOperation`**`()`

see

Syntax **ReadAnalog** *portID*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

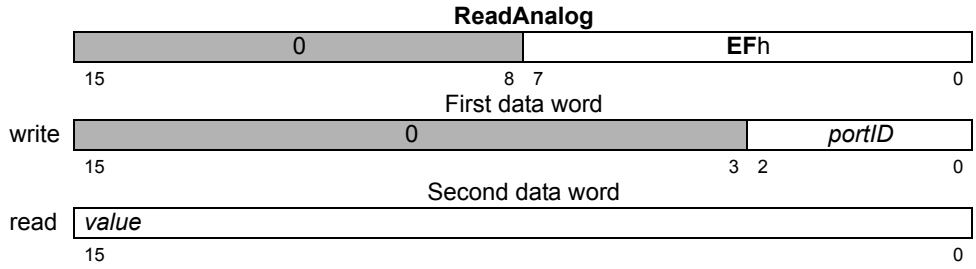
Arguments

Name	Type	Range	Scaling	Units
portID	unsigned 16 bits	0 *to* 7	unity	-

Returned data

	Type	Range	Scaling	Units
value	unsigned 16 bits	0 *to* 2^{16}–1	$100/2^{16}$	% input

Packet Structure



Description **ReadAnalog** returns a 16-bit value representing the voltage presented to the specified analog input. See the product user's guide for more information on analog input and scaling.

Restrictions

C-Motion API

```
PMDresult PMDReadAnalog(PMDAxisInterface axis_intf, PMDuint16 portID,
                        PMDuint16* value)
```

VB-Motion API

```
Dim value as Short
value = MagellanObject.Analog( portID )
```

see

Syntax **ReadBuffer** *bufferID*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

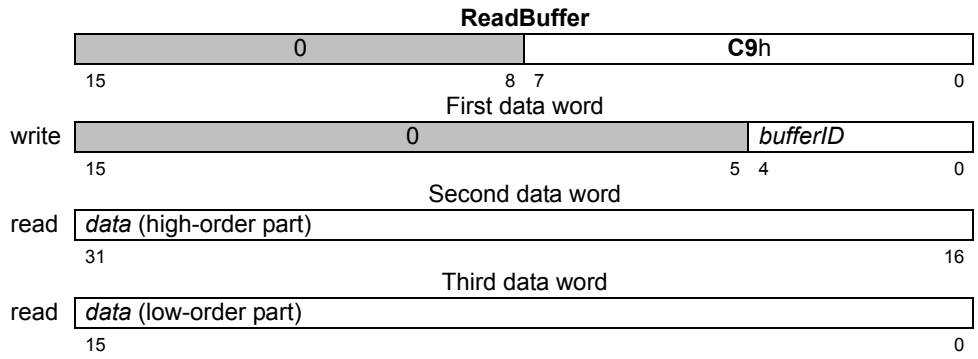
Arguments

Name	Type	Range
bufferID	unsigned 16 bits	0 *to* 31

Returned data

	Type	Range
data	signed 32 bits	-2^{31} *to* $2^{31}-1$

Packet Structure



Description

ReadBuffer returns the 32-bit contents of the location pointed to by the read buffer index in the specified buffer. After the contents have been read, the read index is incremented by 1. If the result is equal to the buffer length (set by **SetBufferLength**), the index is reset to zero (0). The read index is automatically changed at the completion of a trace when in rolling trace mode.

Restrictions

C-Motion API

```
PMDresult PMDReadBuffer(PMDAxisInterface axis_intf, PMDuint16 bufferID,
                        PMDint32* data)
```

VB-Motion API

```
Dim data as Long
Data = MagellanObject.ReadBuffer( bufferID )
```

see **Set/GetBufferReadIndex** (p. 82), **WriteBuffer** (p. 168), **Set/GetBufferStart** (p. 83), **Set/GetBufferLength** (p. 81)

Syntax **ReadIO** *address*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

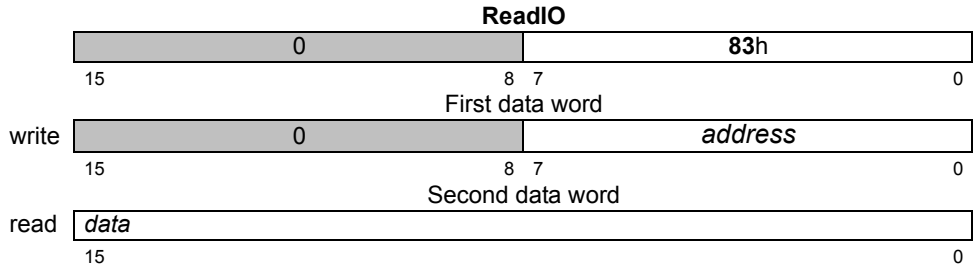
Arguments

Name	Type	Range
address	unsigned 16 bits	0 *to* 255

Returned data

	Type	Range
data	unsigned 16 bits	0 *to* 2^{16}–1

Packet Structure



Description

ReadIO reads one 16-bit word of data from the device at ***address***. The ***address*** is an offset from location 1000h of the motion processor's peripheral device address space.

The format and interpretation of the 16-bit data word are dependent on the user-defined device being addressed. User-defined I/O can be used to implement a number of features, including additional parallel I/O, flash memory for non-volatile configuration information storage, or display devices such as LED arrays.

Restrictions

This command is not available in products without a parallel I/O port.

C-Motion API

```
PMDresult PMDReadIO(PMDAxisInterface axis_intf, PMDuint16 address,
                    PMDuint16* data);
```

VB-Motion API

```
Dim data as Short
data = MagellanObject.IO( address )
```

see **WriteIO** (p. 169)

Syntax **Reset**

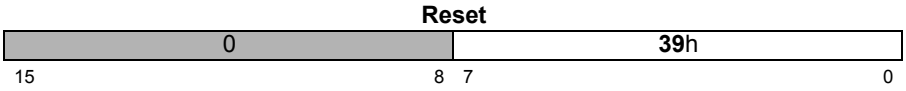
Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments None

Returned data None

Packet Structure



Description

Reset restores the motion processor to its initial condition, setting all motion processor variables to their default values. Most variables are motor-type independent; however several default values depend upon the configured motor type of the axis. The motor-type independent values are listed here.

	Default Value	Buffered
Breakpoints and Interrupts		
Breakpoint 1	0	NO
Breakpoint 2	0	NO
Breakpoint Value 1	0	NO
Breakpoint Value 2	0	NO
Breakpoint Update Mask 1	0Bh	NO
Breakpoint Update Mask 2	0Bh	NO
Interrupt Mask	0	NO
Commutation		
Commutation Mode	motor dependent	NO
Phase Angle	0	NO
Phase Counts	motor dependent	NO
Phase Offset	–1	NO
Phase Prescale	0	NO
Phase Initialize Mode	0	NO
Phase Initialize Time	0	NO
Phase Correction Mode	motor dependent	NO
Current Control		
Currrent Control Mode	0	YES-Current Loop
Current Loop Kp (both A and B loops)	0	YES-Current Loop
Current Loop Ki (both A and B loops)	0	YES-Current Loop
Current Loop Integration Limit (both A and B loops)	0	YES-Current Loop
FOC Kp (both D and Q loops)	0	YES-Current Loop
FOC Ki (both D and Q loops)	0	YES-Current Loop
FOC Integration Limit	0	YES-Current Loop
Holding Motor Limit	32767	NO
Holding Delay	motor dependent	NO

	Default Value	Buffered
Digital Servo Filter		
Position Error Limit	65535	NO
Position Loop Biquad Coeffs	All 0	YES-PositionLoop
Position Loop Biquad Enables	Both 0	YES-Position Loop
Position Loop Kvff	0	YES-Position Loop
Position Loop Kaff	0	YES-Position Loop
Position Loop Kp	0	YES-Position Loop
Position Loop Ki	0	YES-Position Loop
Position Loop Kd	0	YES-Position Loop
Position Loop Integration Limit	0	YES-Position Loop
Position Loop Derivative Time	1	YES-Position Loop
Position Loop Kout	65535	YES-Position Loop
Motor Limit	32767	NO
Motor Bias	0	NO
Motor Command	0	YES-Position Loop
Auxiliary Encoder Source	0	NO
Encoder		
Actual Position	0	NO
Actual Position Units	motor dependent	NO
Capture Source	0	NO
Encoder Modulus	0	NO
Encoder Source	motor dependent	NO
Encoder To Step Ratio	00010001h	NO
Motor Output		
Operating Mode	0033h (Magellan) 0001h (ION)	NO
Active Operating Mode	0033h (Magellan) 0001h (ION)	NO
Output Mode	motor dependent	NO
Motor Type	product dependent	NO
PWM Frequency	motor dependent	NO
Step Range	1	NO
Position Servo Loop Control		
Motion Complete Mode	0	NO
Sample Time	see Notes	NO
Settle Time	0	NO
Settle Window	0	NO
Tracking Window	0	NO
Profile Generation		
Acceleration	0	YES-Trajectory
Deceleration	0	YES-Trajectory
Gear Master	0	NO
Gear Ratio	0	YES-Trajectory
Jerk	0	YES-Trajectory
Position	0	YES-Trajectory
Profile Mode	0	YES-Trajectory
Start Velocity	0	NO
Stop Mode	0	YES-Trajectory
Velocity	0	YES-Trajectory

	Default Value	Buffered
RAM Buffer		
Buffer Length	0-Magellan 0180h-ION	NO
Buffer Read Index	0	NO
Buffer Start	0	NO
Buffer Write Index	0	NO
Safety		
Positive Limit Event Action	8	NO
Negative Limit Event Action	8	NO
Motion Error Event Action	motor dependent	NO
Current Foldback Event Action	7	NO
OvervoltageThreshold	see *ION Digital Drive User's Manual*	NO
Undervoltage Threshold	see *ION Digital Drive User's Manual*	NO
OvertemperatureThreshold	17920	NO
FaultOut Mask	0600h	NO
Continuous Current Limit	see *ION Digital Drive User's Manual*	
Energy Limit	see *ION Digital Drive User's Manual*	
Status Registers and AxisOut Indicator		
AxisOut Source Axis	0	NO
AxisOut Register	0	NO
AxisOut Selection Mask	0	NO
AxisOut Sense Mask	0	NO
Signal Sense	motor dependent	NO
Traces		
Trace Mode	0	NO
Trace Period	1	NO
Trace Start	0	NO
Trace Stop	0	NO
Trace Variables	all are 0	NO
Miscellaneous		
Update Mask	0Bh	NO
CAN Mode	C000h (see Notes)	NO
Serial Port Mode 232	0004h (see Notes)	NO
Serial Port Mode 485	0084h (see Notes)	NO

The motor-type dependent default values are listed in the following tables.

Variable	DC Brush	Brushless DC (3 phase)	Brushless DC (2 phase)
Actual Position Units	0	0	0
Commutation Mode	-	0	0
Encoder Source	0	0	0
Motion Error Event Action	5	5	5
Output Mode	1-Magellan 2-ION	2	2
Phase Correction Mode	-	1	1
PWM Frequency (kHz)	20	20	20
SPI Mode	0	-	-
Phase Counts	-	1	1
Holding Delay	-	-	-
Signal Sense	0	0	0

Variable	Microstepping (3 phase)	Microstepping (2 phase)	Pulse & Direction
Actual Position Units	1	1	1
Commutation Mode	0	0	-
Encoder Source	2	2	3
Motion Error Event Action	0	0	0
Output Mode	2	1-Magellan 2-ION	-
Phase Correction Mode	-	-	-
PWM Frequency (kHz)	20	80-Magellan 20-ION	-
SPI Mode	-	-	-
Phase Counts	256	256	-
Holding Delay	32767	32767	20
Signal Sense	0	0	0800h

Notes

All axes supported by the motion processor are enabled at reset.

In some products, CAN Mode and Serial Port Mode defaults are defined at reset by a parallel bus read.

In some products, the CAN Mode and Serial Port Mode 485 reset defaults can be over-ridden using the **SetDefault** command.

See **Set/GetSampleTime** (p. 140) for more information regarding SampleTime.

Restrictions

The typical time before the device is ready for communication after a reset is 20ms for Magellan products, and 100ms for ION products.

The MC55110 and the MC58110 have a maximum Step Range of 100 ksteps/sec, which cannot be changed.

Not all of the listed variables are available on all products. See the product user's guide.

C-Motion API

```
PMDresult PMDReset(PMDAxisInterface axis_intf)
```

VB-Motion API

```
MagellanObject.Reset()
```

see

SetDefault (p. 96)

Syntax **ResetEventStatus** *axis mask*

Motor Types

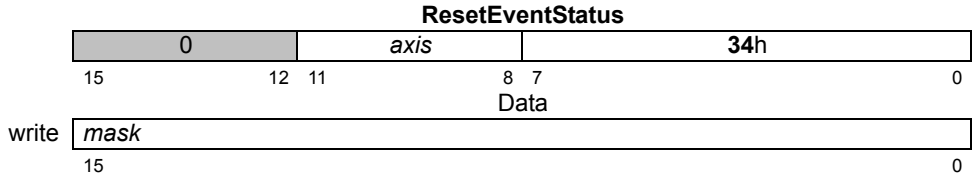
DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
mask	*Motion Complete*	FFFEh
	Wrap-around	FFFDh
	Breakpoint 1	FFFBh
	Capture Received	FFF7h
	Motion Error	FFEFh
	Positive Limit	FFDFh
	Negative Limit	FFBFh
	Instruction Error	FF7Fh
	Disable	FEFFh
	Overtemperature Fault	FDFFh
	Bus Voltage Fault	FBFFh
	Commutation Error	F7FFh
	Current Foldback	EFFFh
	Breakpoint 2	BFFFh

Returned data None

Packet Structure



Description

ResetEventStatus clears (sets to 0), for the specified ***axis***, each bit in the Event Status register that has a value of 0 in the ***mask*** sent with this command. All other Event Status register bits (bits that have a mask value of 1) are unaffected.

Events that cause changes in operating mode or trajectory require, in general, that the corresponding bit in Event Status be cleared prior to returning to operation. That is, prior to restoring the operating mode (in cases where the event caused a change in it) or prior to performing another trajectory move (in cases where the event caused a trajectory stop). The one exception to this is ***Motion Error***, which is not required to be cleared if the event action for it includes disabling of the position loop.

Restrictions Not all bits in **ResetEventStatus** are supported in some products. See the product user's manual.

C-Motion API

```
PMDresult PMDResetEventStatus(PMDAxisInterface axis_intf,
                              PMDuint16 status)
```

VB-Motion API

```
MagellanAxis.ResetEventStatus( mask )
```

see **GetEventStatus** (p. 39)

Syntax **RestoreOperatingMode** *axis*

Motor Types

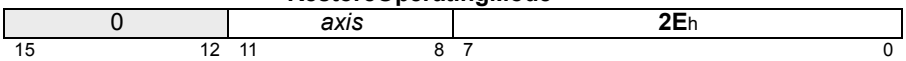
DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

Packet Structure

RestoreOperatingMode



Description **RestoreOperatingMode** is used to command the ***axis*** to return to its static operating mode. It should be used when the active operating mode has changed due to actions taken from safety events or other programmed events. Calling **RestoreOperatingMode** will re-enable all loops that were disabled as a result of events.

Restrictions Before using **RestoreOperatingMode** to return to the static operating mode, the event status bits should all be cleared. If a bit in event status that caused a change in operating mode is not cleared, this command will return an error. An exception to this is Motion Error, which does not have to be cleared prior to restoring the operating mode.

Though **RestoreOperatingMode** will re-enable the trajectory generator (if it was disabled as a result of an event action), it will not resume a move. This must be done through an **Update** or **MultiUpdate**.

C-Motion API `PMDresult` **`PMDRestoreOperatingMode`**`(PMDAxisInterface` *`axis_intf`*`)`

VB-Motion API **`MagellanAxis.RestoreOperatingMode`**`()`

see **GetActiveOperatingMode** (p. 25), **Set/GetOperatingMode** (p. 122), **Set/GetEventAction** (p. 102), **Set/GetBreakpoint** (p. 74)

SetAcceleration buffered 90h
GetAcceleration 4Ch

Syntax

SetAcceleration *axis acceleration*
GetAcceleration *axis*

Motor Types

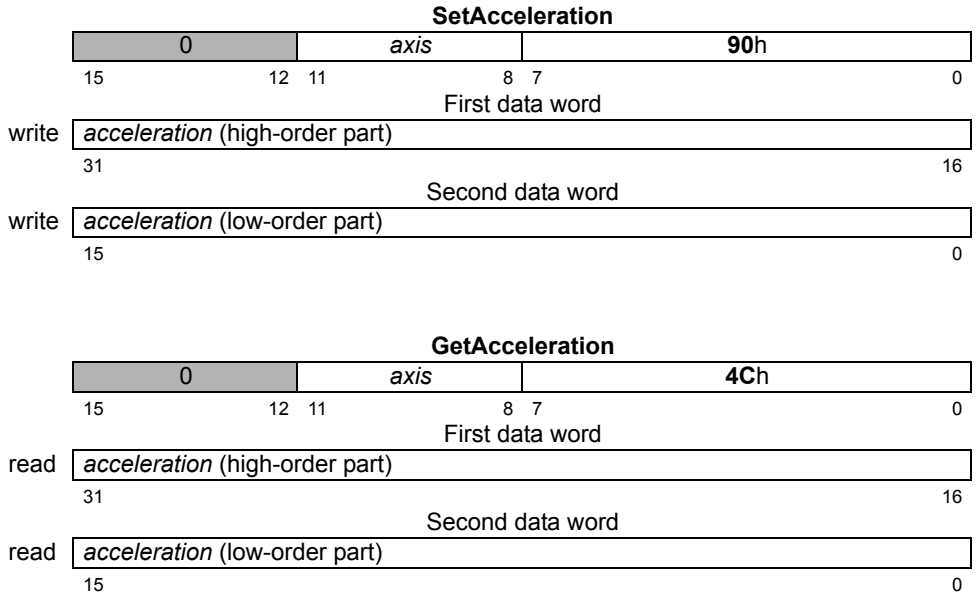
DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

	Type	Range	Scaling	Units
acceleration	unsigned 32 bits	0 *to* $2^{31}-1$	$1/2^{16}$	counts/cycle^2 microsteps/cycle^2

Packet Structure



Description

SetAcceleration loads the maximum acceleration buffer register for the specified *axis*. This command is used with the Trapezoidal, Velocity Contouring, and S-curve profiling modes.

GetAcceleration reads the maximum acceleration buffer register.

Scaling example: To load a value of 1.750 counts/cycle^2, multiply by 65,536 (given 114,688) and load the resultant number as a 32-bit number, giving 0001 in the high word and C000h in the low word. Values returned by **GetAcceleration** must correspondingly be divided by 65,536 to convert to units of counts/cycle^2 or steps/cycle^2.

Restrictions

SetAcceleration may not be issued while an axis is in motion with the S-curve profile.

SetAcceleration is not valid in Electronic Gear profile mode.

SetAcceleration is a buffered command. The value set using this command will not take effect until the next **Update** or **MultiUpdate** command, with the Trajectory Update bit set in the update mask.

C-Motion API

```
PMDresult PMDSetAcceleration(PMDAxisInterface axis_intf,
                             PMDuint32 acceleration)
PMDresult PMDGetAcceleration(PMDAxisInterface axis_intf,
                             PMDuint32* acceleration)
```

VB-Motion API

```
Dim acceleration as Long
MagellanAxis.Acceleration = acceleration
acceleration = MagellanAxis.Acceleration
```

see

Set/GetDeceleration (p. 95), **Set/GetJerk** (p. 115), **Set/GetPosition** (p. 133), **Set/GetVelocity** (p. 165), **MultiUpdate** (p. 55), **Update** (p. 167)

SetActualPosition 4Dh
GetActualPosition 37h

Syntax

SetActualPosition *axis position*
GetActualPosition *axis*

Motor Types

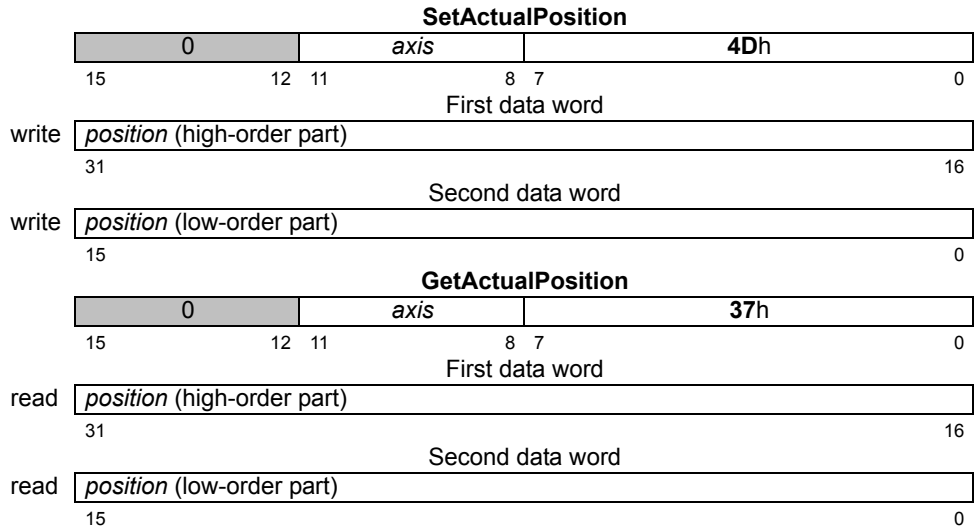
DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

	Type	Range	Scaling	Units
position	signed 32 bits	-2^{31} *to* $2^{31}-1$	unity	counts microsteps

Packet Structure



Description

SetActualPosition loads the position register (encoder position) for the specified ***axis***. At the same time, the commanded position is replaced by the loaded value minus the position error. This prevents a servo "bump" when the new axis position is established. The destination position (see **SetPosition** (p. 133)) is also modified by this amount so that no trajectory motion will occur when the **Update** instruction is issued. In effect, this instruction establishes a new reference position from which subsequent positions can be calculated. It is commonly used to set a known reference position after a homing procedure.

Note: For axes configured as pulse & direction or microstepping motor types, actual position units determines if the position is specified and returned in units of counts or steps. Additionally, for these motor types, the position error is zeroed when the **SetActualPosition** command is sent. **SetActualPosition** takes effect immediately, it is not buffered.

GetActualPosition reads the contents of the encoder's actual position register. This value will be accurate to within one cycle (as determined by **Set/GetSampleTime**).

Restrictions

C-Motion API

```
PMDresult PMDSetActualPosition(PMDAxisInterface axis_intf,
                               PMDint32 position)
PMDresult PMDGetActualPosition(PMDAxisInterface axis_intf,
                               PMDint32* position)
```

VB-Motion API

```
Dim position as Long
MagellanAxis.ActualPosition = position
position = MagellanAxis.ActualPosition
```

see **GetPositionError** (p. 45), **Set/GetActualPositionUnits** (p. 70), **AdjustActualPosition** (p. 20)

SetActualPositionUnits BEh
GetActualPositionUnits BFh

Syntax

SetActualPositionUnits *axis mode*
GetActualPositionUnits *axis*

Motor Types

		Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
mode	*Counts*	0
	Steps	1

Packet Structure

SetActualPositionUnits

	0	*axis*	**BE**h
	15 12	11 8	7 0
	Data		
write	0		*mode*
	15		1 0

GetActualPositionUnits

	0	*axis*	**BF**h
	15 12	11 8	7 0
	Data		
read	0		*mode*
	15		1 0

Description

SetActualPositionUnits determines the units used by the **Set/GetActualPosition**, **AdjustActualPosition** and **GetCaptureValue** for the specified *axis*. It also affects the trace variable Actual Position. When set to *Counts*, position units are in encoder counts. When set to *Steps*, position units are in steps/microsteps. The step position is calculated using the ratio as set by the **SetEncoderToStepRatio** command.

GetActualPositionUnits returns the position units for the specified *axis*.

Restrictions

The trace variable, capture value, is not affected by this command. The value is always in counts.

C-Motion API

```
PMDresult PMDSetActualPositionUnits(PMDAxisInterface axis_intf,
                                    PMDuint16 mode)
PMDresult PMDGetActualPositionUnits(PMDAxisInterface axis_intf,
                                    PMDuint16* mode)
```

VB-Motion API

```
Dim mode as Short
MagellanAxis.ActualPositionUnits = mode
mode = MagellanAxis.ActualPositionUnits
```

see

Set/GetActualPosition (p. 68), **Set/GetEncoderToStepRatio** (p. 101), **AdjustActualPosition** (p. 20), **GetCaptureValue** (p. 30), **Set/GetTraceVariable** (p. 159)

Syntax

SetAuxiliaryEncoderSource *axis mode auxiliaryAxis*
GetAuxiliaryEncoderSource *axis*

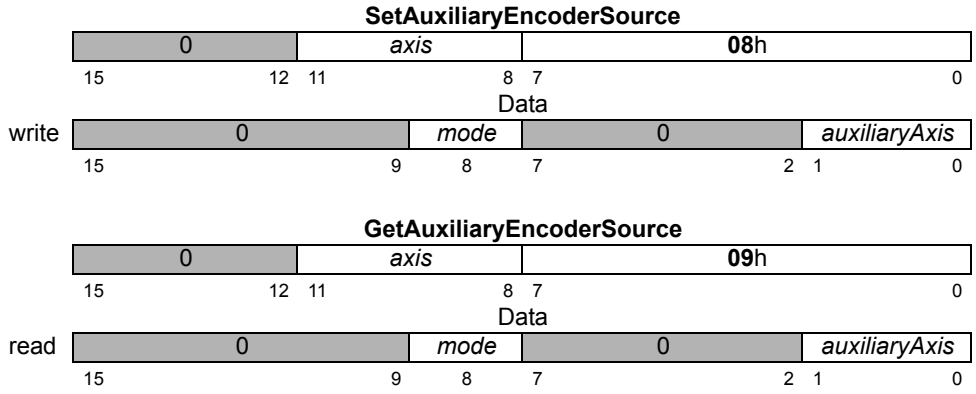
Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
mode	*Disable*	0
	Enable	1
auxiliaryAxis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

Packet Structure



Description

SetAuxiliaryEncoderSource controls the motion processor's dual encoder loop feature. The ***mode*** enables or disables the secondary encoder loop for ***axis***. The ***auxilaryAxis*** selects which axis encoder input is to be interpreted as the damping term (Kd) of the servo equation for ***axis***. To determine the actual position of the auxiliary encoder, use **GetActualPosition**(*auxiliaryAxis*). The auxiliary axis encoder input is used for commutation of brushless DC motors. The **SetPhaseOffset**, **SetPhaseAngle**, and **SetPhaseCounts** commands should still be applied to the main axis, even when dual encoder loop is enabled.

Restrictions

To avoid a potentially unstable operating condition in dual loop mode, the auxiliary encoder should have resolution greater than or equal to that of the main encoder.

C-Motion API

```
PMDresult PMDSetAuxiliaryEncoderSource(PMDAxisInterface axis_intf,
                                       PMDuint8 mode,
                                       PMDAxis auxiliaryAxis)
PMDresult PMDGetAuxiliaryEncoderSource(PMDAxisInterface axis_intf,
                                       PMDuint8* mode,
                                       PMDAxis* auxiliaryAxis)
```

VB-Motion API

```
MagellanAxis.AuxiliaryEncoderSourceSet( [in] mode, [in] auxiliaryAxis )
MagellanAxis.AuxiliaryEncoderSourceGet( [out] mode, [out] auxiliaryAxis )
```

SetAxisOutMask 45h
GetAxisOutMask 46h

Syntax

SetAxisOutMask *axis sourceAxis sourceRegister selectionMask senseMask*
GetAxisOutMask *axis*

Motor Types

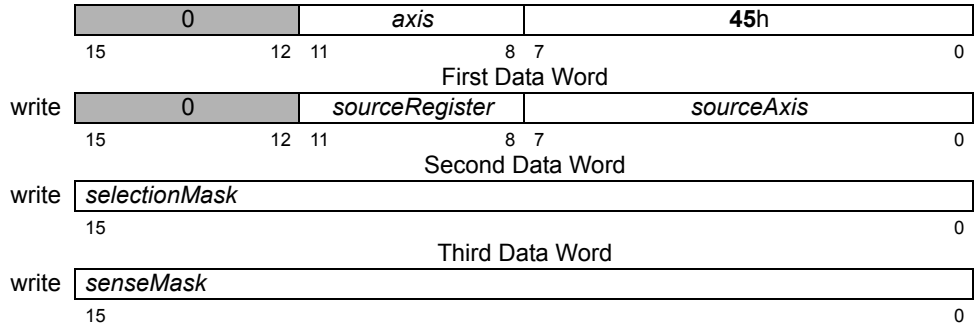
DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

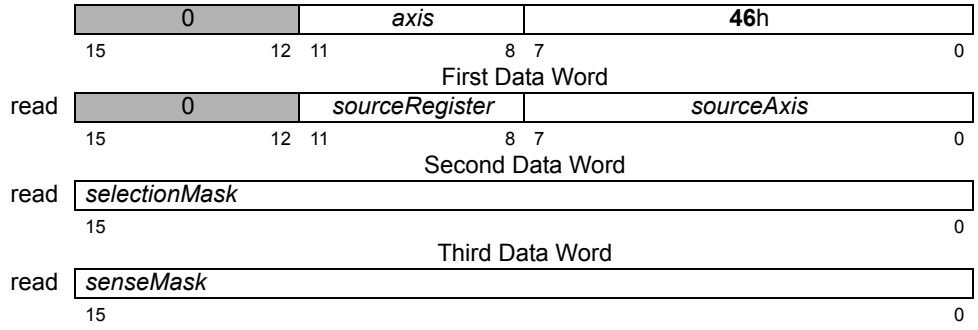
Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
sourceAxis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
sourceRegister	*Disabled*	0
	Event Status	1
	Activity Status	2
	Signal Status	3
	Drive Status	4
selectionMask	see below	bitmask
senseMask	see below	bitmask

Packet Structure

SetAxisOutMask



GetAxisOutMask



Description

SetAxisOutMask configures what will drive the AxisOut pin of the ***axis***. The ***sourceRegister*** and ***sourceAxis*** arguments specify which register, from which axis, will be used to drive AxisOut of the specified axis. For each bit in the ***selectionMask*** that is set to 1, the corresponding bit of the specified ***sourceRegister*** is selected to set AxisOut active. The ***senseMask*** bit determines which state of each bit causes AxisOut to be active—a zero (0) in the ***senseMask*** means that a 0 in the corresponding bit will cause AxisOut to be active, and a 1 in the ***senseMask*** means that a 1 in the corresponding bit will cause AxisOut to be active. If multiple bits are selected in the ***sourceRegister***, AxisOut will be active if any of the selected bits, combined with their sense, require it to be. The following table shows the available bits in each register.

Bit	Event Status Register	Activity Status Register	Signal Status Register	Drive Status Register
0	Motion Complete	Phasing Initialized	Encoder A	
1	Wrap-around	At Maximum Velocity	Encoder B	In Foldback
2	Breakpoint 1	Tracking	Encoder Index	Overtemperature
3	Position Capture		Capture Input	
4	Motion Error		Positive Limit	In Holding
5	Positive Limit		Negative Limit	Overvoltage
6	Negative Limit		AxisIn	Undervoltage
7	Instruction Error	Axis Settled	Hall Sensor A	
8	Disable	Motor Mode	Hall Sensor B	
9	Overtemperature Fault	Position Capture	Hall Sensor C	
0Ah	Bus Voltage Fault	In Motion		
0Bh	Commutation Error	In Positive Limit		
0Ch	Current Foldback	In Negative Limit		
0Dh			/Enable Input	
0Eh	Breakpoint 2		FaultOut	
0Fh				

For example, assume it is desired to have the AxisOut pin of ***Axis1*** driven active whenever motion complete of ***Axis2*** is 1, or commutation error of ***Axis2*** is 0. In this case, ***axis*** would be 0 (***Axis1***), ***sourceAxis*** would be 1 (***Axis2***), ***sourceRegister*** would be 1 (***Event Status***), ***selectionMask*** would be 0801h (commutation error and motion complete) and ***senseMask*** would be 0001h.

GetAxisOutMask returns the mapping of the AxisOut pin of ***axis***.

Restrictions

Depending on the product features, some bits may not be supported. See the product user's guide.

C-Motion API

```
PMDresult PMDSetAxisOutMask(PMDAxisInterface axis_intf,
                            PMDAxis sourceAxis,
                            PMDuint8 sourceRegister,
                            PMDuint16 selectionMask,
                            PMDuint16 senseMask)
PMDresult PMDGetAxisOutMask(PMDAxisInterface axis_intf,
                            PMDAxis* sourceAxis,
                            PMDuint8* sourceRegister,
                            PMDuint16* selectionMask,
                            PMDuint16* senseMask)
```

VB-Motion API

```
MagellanAxis.AxisOutMaskSet( [in] sourceAxis,
                             [in] sourceRegister,
                             [in] selectionMask,
                             [in] senseMask )
MagellanAxis.AxisOutMaskGet( [out] sourceAxis,
                             [out] sourceRegister,
                             [out] selectionMask,
                             [out] senseMask )
```

see

SetBreakpoint D4h
GetBreakpoint D5h

Syntax

SetBreakpoint *axis breakpointID sourceAxis action trigger*
GetBreakpoint *axis breakpointID*

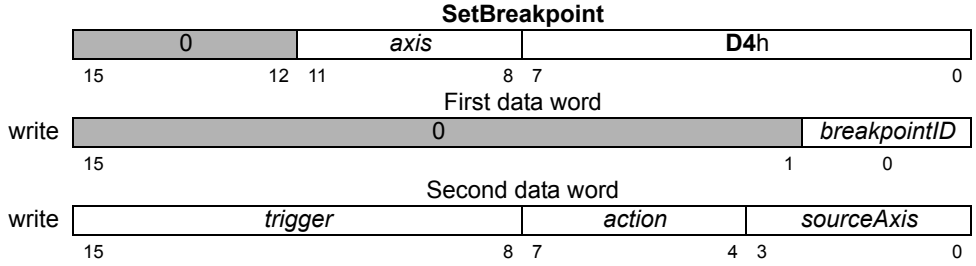
Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

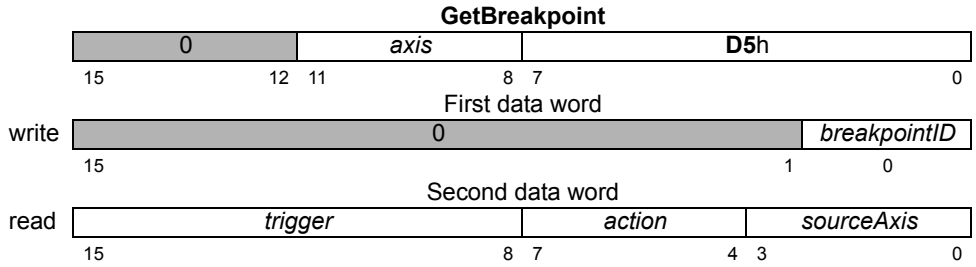
Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
breakpointID	*Breakpoint1*	0
	Breakpoint2	1
sourceAxis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
action	*None*	0
	Update	1
	Abrupt Stop	2
	Smooth Stop	3
	— (Reserved)	4
	Disable Position Loop & Higher Modules	5
	Disable Current Loop & Higher Modules	6
	Disable Motor Output & Higher Modules	7
	Abrupt Stop with Position Error Clear	8
trigger	*None*	0
	Greater Or Equal Commanded Position	1
	Lesser Or Equal Commanded Position	2
	Greater Or Equal Actual Position	3
	Lesser Or Equal Actual Position	4
	Commanded Position Crossed	5
	Actual Position Crossed	6
	Time	7
	Event Status	8
	Activity Status	9
	Signal Status	Ah
	Drive Status	Bh

Packet Structure



Packet Structure (cont.)



Description

SetBreakpoint establishes a breakpoint for the specified ***axis*** to be triggered by a condition or event on ***sourceAxis***, which may be the same as or different from ***axis***. Up to two concurrent breakpoints can be set for each axis, each of which may have its own breakpoint type and comparison value. The ***breakpointID*** field specifies which breakpoint the **SetBreakpoint** and **GetBreakpoint** commands will address.

The six position breakpoints and the ***Time*** breakpoint are threshold-triggered; the breakpoint occurs when the indicated value reaches or crosses a threshold. The status breakpoints are level-triggered; the breakpoint occurs when a specific bit or combination of bits in the indicated status register changes state. Thresholds and bit specifications are both set by the **SetBreakpointValue** instruction.

The ***action*** determines what the motion processor does when the breakpoint occurs, as follows:

Action	Description
None	No action
Update	Transfer the double buffered registers specified by the Breakpoint Update Mask into the active registers.
Abrupt Stop	Causes an instantaneous halt of the trajectory generator. Trajectory velocity is zeroed.
Smooth Stop	Causes a smooth stop to occur at the active deceleration rate.
Abrupt Stop with Position Error Clear	Abrupt stop of the trajectory, and additionally zero the position error to the servo.
Disable Position Loop & Higher Modules	Disables Trajectory generator and position loop modules.
Disable Current Loop & Higher Modules	Disables Trajectory generator, position loop, and current loop modules.
Disable Motor Output & Higher Modules	Disables all modules, including the motor output.

GetBreakpoint returns the ***trigger***, ***action***, and ***sourceAxis*** for the specified breakpoint (1 or 2) of the indicated ***axis***. When a breakpoint occurs, the trigger value will be reset to zero (0). The ***Commanded Position Crossed*** and the ***Actual Position Crossed*** triggers are converted to one of the position trigger types 1–4, depending on the current position when the command is issued.

Restrictions

Always load the breakpoint comparison value (**SetBreakpointValue** command) before setting a new breakpoint condition (**SetBreakpoint** command). Failure to do so will likely result in unexpected behavior.

Breakpoint trigger options may be limited depending on the resources of the ***sourceAxis***. See the product user's guide.

SetBreakpoint (cont.) D4h
GetBreakpoint D5h

C-Motion API

```
PMDresult PMDSetBreakpoint(PMDAxisInterface axis_intf,
                           PMDuint16 breakpointID, PMDAxis sourceAxis,
                           PMDuint8 action, PMDuint8 trigger)
PMDresult PMDGetBreakpoint(PMDAxisInterface axis_intf,
                           PMDuint16 breakpointID, PMDAxis* sourceAxis,
                           PMDuint8* action, PMDuint8* trigger)
```

VB-Motion API

```
MagellanAxis.BreakpointSet( [in] breakpointID,
                            [in] sourceAxis,
                            [in] action,
                            [in] trigger )
MagellanAxis.BreakpointGet( [in] breakpointID,
                            [out] sourceAxis,
                            [out] action,
                            [out] trigger )
```

see

Set/GetBreakpointValue (p. 79), **Set/GetBreakpointUpdateMask** (p. 77)

Syntax

SetBreakpointUpdateMask *axis breakPointID mask*
GetBreakpointUpdateMask *axis breakPointID*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

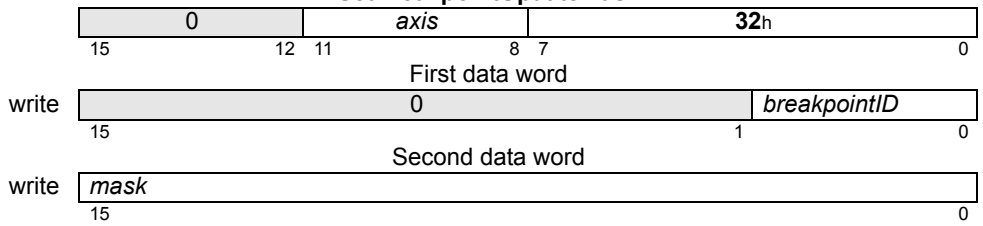
Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
breakpointID	*Breakpoint1*	0
	Breakpoint2	1

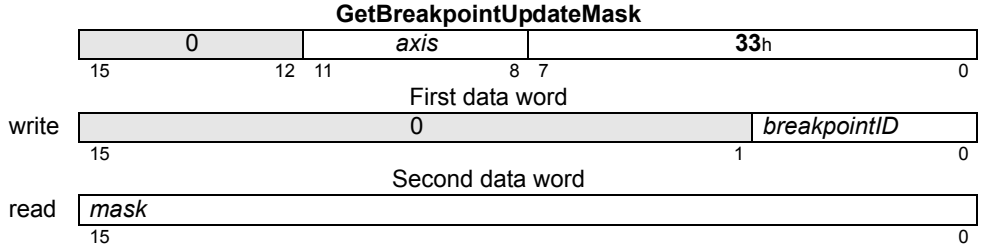
	Type	Scaling
mask	unsigned 16 bit	bitmask

Packet Structure

SetBreakpointUpdateMask



GetBreakpointUpdateMask



Description

SetBreakpointUpdateMask configures what loops are updated upon the update action of a breakpoint. If the bitmask for a given loop is set in the ***mask***, the operating parameters for that loop will be updated from the buffered values when the breakpoint is hit, and update is the breakpoint action. Each breakpoint has its own update mask. The bitmask encoding is given below.

Name	Bit(s)	Description
Trajectory	0	Set to 1 to update trajectory from buffered parameters.
Position Loop	1	Set to 1 to update position loop from buffered parameters.
—	2	Reserved
Current Loop	3	Set to 1 to update current loop from buffered parameters.
—	4–15	Reserved

For example, if the update mask for breakpoint 1 is set to hexadecimal 0001h, and the action for breakpoint 1 is set to update, the trajectory for the given ***axis*** will be updated from its buffered parameters when breakpoint 1 is hit.

Description (cont.)

The Current Loop bit applies regardless of the active current control mode. When it is set, a breakpoint action of update will update either the active FOC parameters or the active digital current loop parameters, depending on which Current Control mode is active.

GetBreakpointUpdateMask gets the update mask for the indicated breakpoint.

Restrictions

The Current Loop bit is only valid for products that include a current loop.

C-Motion API

```
PMDresult PMDSetBreakpointUpdateMask(PMDAxisInterface axis_intf,
                                     PMDuint16 breakPointID,
                                     PMDuint16 mask)
PMDresult PMDGetBreakpointUpdateMask(PMDAxisInterface axis_intf,
                                     PMDuint16 breakPointID,
                                     PMDuint16* mask)
```

VB-Motion API

```
Dim mask as Short
MagellanAxis.BreakpointUpdateMask( breakpointID ) = mask
mask = MagellanAxis.BreakpointUpdateMask( breakpointID )
```

see

Set/GetBreakpoint (p. 74), **Set/GetUpdateMask** (p. 163)

Syntax

SetBreakpointValue *axis breakpointID value*
GetBreakpointValue *axis breakpointID*

Motor Types

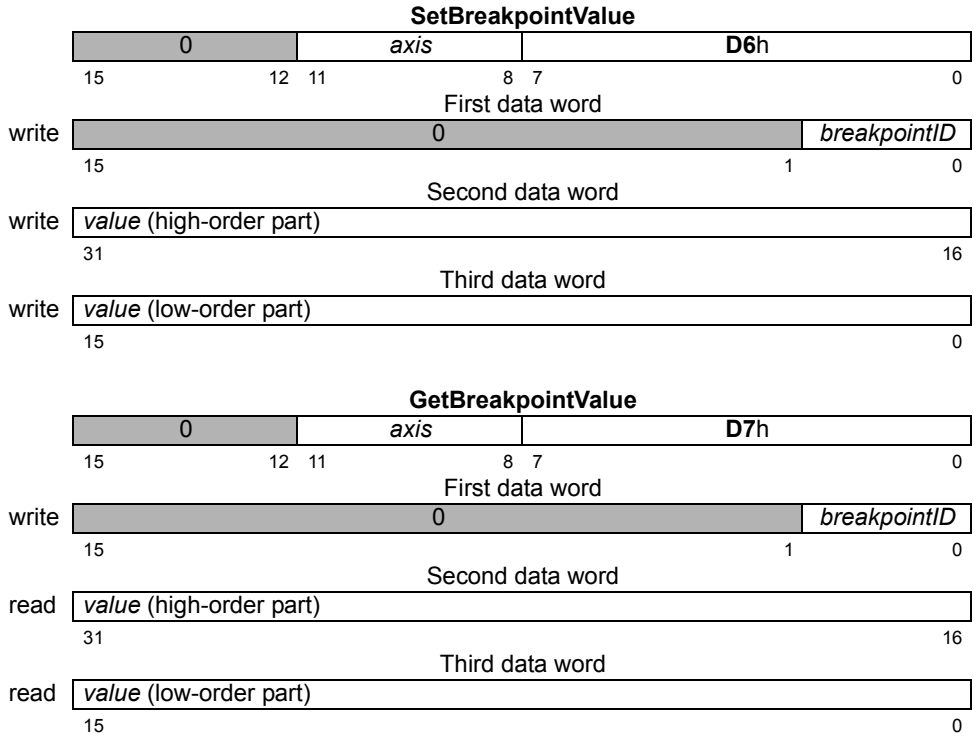
DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
breakpointID	*Breakpoint1*	0
	Breakpoint2	1

value (see below)

Packet Structure



Description

SetBreakpointValue sets the breakpoint comparison value for the specified ***axis***. For the position and time breakpoints, this is a threshold comparison value.

SetBreakpointValue (cont.) D6h
GetBreakpointValue D7h

Description (cont.)

The *value* parameter is interpreted according to the trigger condition for the selected breakpoint; see **SetBreakpoint** (p. 74). The data format for each trigger condition is as follows:

Breakpoint Trigger	Value Type	Range	Units
Greater Or Equal Commanded Position	signed 32-bit	-2^{31} to $2^{31}-1$	counts
Lesser Or Equal Commanded Position	signed 32-bit	-2^{31} to $2^{31}-1$	counts
Greater Or Equal Actual Position	signed 32-bit	-2^{31} to $2^{31}-1$	counts
Lesser Or Equal Actual Position	signed 32-bit	-2^{31} to $2^{31}-1$	counts
Commanded Position Crossed	signed 32-bit	-2^{31} to $2^{31}-1$	counts
Actual Position Crossed	signed 32-bit	-2^{31} to $2^{31}-1$	counts
Time	unsigned 32-bit	0 to $2^{32}-1$	cycles
Event Status	2 word mask	-	boolean status values
Activity Status	2 word mask	-	boolean status values
Signal Status	2 word mask	-	boolean status values
Drive Status	2 word mask	-	boolean status values

For level-triggered breakpoints, the high-order part of *value* is the selection mask, and the low-order word is the sense mask. For each selection bit that is set to 1, the corresponding bit of the specified status register is conditioned to cause a breakpoint when it changes state. The sense mask bit determines which state causes the break. If it is 1, the corresponding status register bit will cause a break when it is set to 1. If it is 0, the status register bit will cause a break when it is set to 0.

For example, assume it is desired that the breakpoint type will be set to Event Status and that a breakpoint should be recognized whenever the motion complete bit (bit 0 of Event Status register) is set to 1, or the commutation error bit (bit 11 of Event Status register) is set to 0. In this situation the high and low words for value would be high word: 0801h and low word: 0001h.

GetBreakpointValue returns the breakpoint value for the specified *breakpointID*.

Two completely separate breakpoints are supported, each of which may have its own breakpoint type and comparison value. The *breakpointID* field specifies which breakpoint the **SetBreakpointValue** and **GetBreakpointValue** commands will address.

Restrictions

Always load the breakpoint comparison value (**SetBreakpointValue** command) before setting a new breakpoint condition (**SetBreakpoint** command). Failure to do so will likely result in unexpected behavior.

Depending on the product features, not all bits of all registers are supported. See the product user's guide.

C-Motion API

```
PMDresult PMDSetBreakpointValue(PMDAxisInterface axis_intf,
                                PMDuint16 breakpointID,
                                PMDint32 value)
PMDresult PMDGetBreakpointValue(PMDAxisInterface axis_intf,
                                PMDuint16 breakpointID,
                                PMDint32* value)
```

VB-Motion API

```
Dim value as Long
MagellanAxis.BreakpointValue( breakpointID ) = value
value = MagellanAxis.BreakpointValue( breakpointID )
```

see

Set/GetBreakpoint (p. 74)

Syntax

SetBufferLength *bufferID length*
GetBufferLength *bufferID*

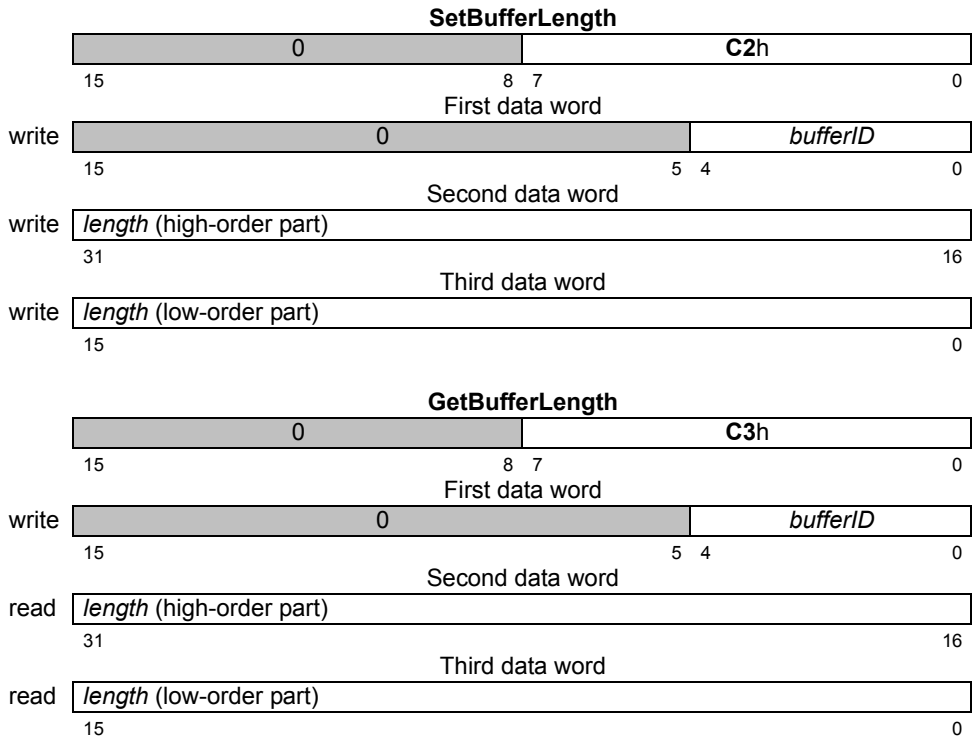
Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Type	Range
bufferID	unsigned 16 bits	0 *to* 31
length	unsigned 32 bits	1 *to* $2^{30} - 1$

Packet Structure



Description

SetBufferLength sets the ***length***, in numbers of 32-bit elements, of the buffer in the memory block identified by ***bufferID***.

Note: The **SetBufferLength** command resets the buffers read and write indexes to 0.

The **GetBufferLength** command returns the ***length*** of the specified buffer.

Restrictions

The buffer length plus the buffer start address cannot exceed the memory size of the product. See the product user's guide.

C-Motion API

```
PMDresult PMDSetBufferLength(PMDAxisInterface axis_intf,
                             PMDuint16 bufferID, PMDuint32 length)
PMDresult PMDGetBufferLength(PMDAxisInterface axis_intf,
                             PMDuint16 bufferID, PMDuint32* length)
```

VB-Motion API

```
Dim length as Long
MagellanObject.BufferLength( bufferID ) = length
length = MagellanObject.BufferLength( bufferID )
```

see

Set/GetBufferReadIndex (p. 82), **Set/GetBufferStart** (p. 83), **Set/GetBufferWriteIndex** (p. 84)

SetBufferReadIndex C6h
GetBufferReadIndex C7h

Syntax

SetBufferReadIndex *bufferID index*
GetBufferReadIndex *bufferID*

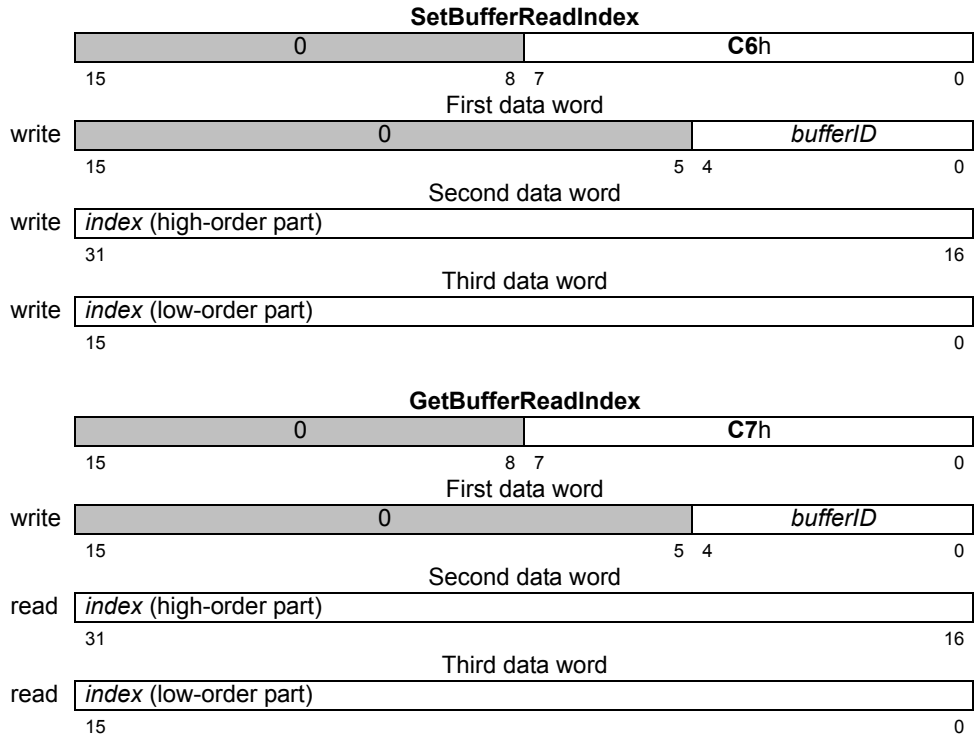
Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Type	Range	Scaling	Units
bufferID	unsigned 16 bits	0 *to* 31	unity	-
index	unsigned 32 bits	0 *to* buffer length - 1	unity	double words

Packet Structure



Description

SetBufferReadIndex sets the address of the read *index* for the specified **bufferID**.

GetBufferReadIndex returns the current read *index* for the specified **bufferID**.

Restrictions

If the read index is set to an address beyond the length of the buffer, the command will not be executed and will return host I/O error code 7, buffer bound exceeded.

C-Motion API

```
PMDresult PMDSetBufferReadIndex(PMDAxisInterface axis_intf,
                                PMDuint16 bufferID,
                                PMDuint32 index)
PMDresult PMDGetBufferReadIndex(PMDAxisInterface axis_intf,
                                PMDuint16 bufferID,
                                PMDuint32* index)
```

VB-Motion API

```
Dim index as Long
MagellanObject.BufferReadIndex( bufferID ) = index
index = MagellanObject.BufferReadIndex( bufferID )
```

see

Set/GetBufferLength (p. 81), **Set/GetBufferStart** (p. 83), **Set/GetBufferWriteIndex** (p. 84)

Syntax

SetBufferStart *bufferID address*
GetBufferStart *bufferID*

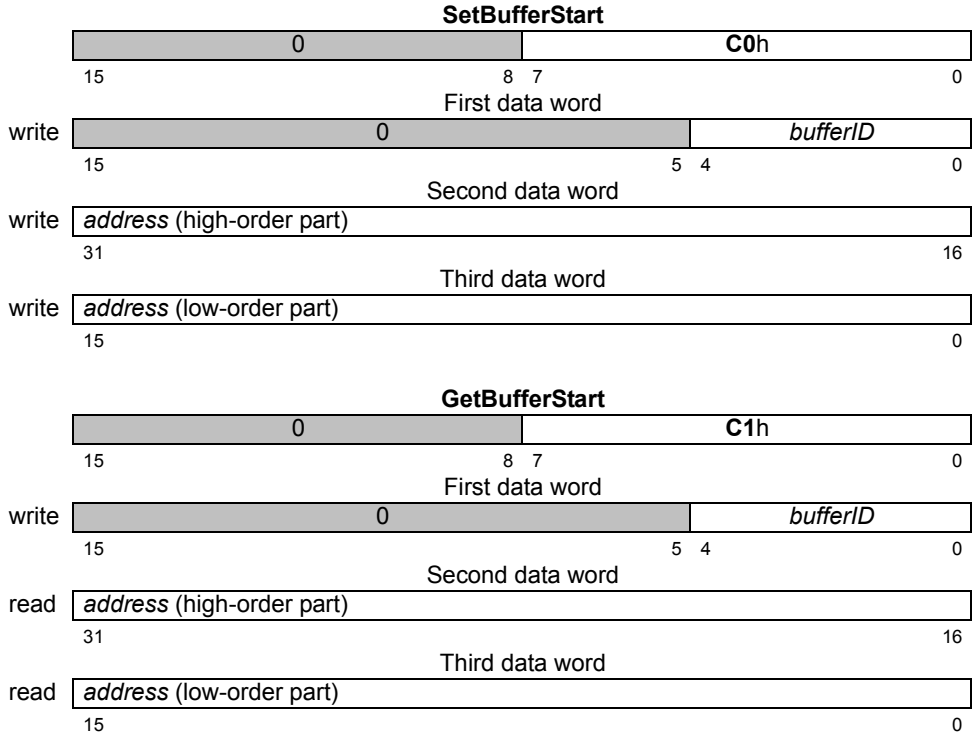
Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Type	Range	Units
bufferID	unsigned 16 bits	0 *to* 31	-
address	unsigned 32 bits	0 *to* $2^{31} - 1$	double words

Packet Structure



Description

SetBufferStart sets the starting ***address*** for the specified buffer, in double-words, of the buffer in the memory block identified by ***bufferID***.

Note: The **SetBufferStart** command resets the buffers read and write indexes to 0.

The **GetBufferStart** command returns the starting ***address*** for the specified ***bufferID***.

Restrictions

The buffer start address plus the buffer length cannot exceed the memory size of the product. See the product user's guide.

C-Motion API

```
PMDresult PMDSetBufferStart(PMDAxisInterface axis_intf, PMDuint16 bufferID,
                            PMDuint32 address)
PMDresult PMDGetBufferStart(PMDAxisInterface axis_intf, PMDuint16 bufferID,
                            PMDuint32* address)
```

VB-Motion API

```
Dim address as Long
MagellanObject.BufferStart( bufferID ) = address
address = MagellanObject.BufferStart( bufferID )
```

see

Set/GetBufferLength (p. 81), **Set/GetBufferReadIndex** (p. 82), **Set/GetBufferWriteIndex** (p. 84)

SetBufferWriteIndex C4h
GetBufferWriteIndex C5h

Syntax

SetBufferWriteIndex *bufferID index*
GetBufferWriteIndex *bufferID*

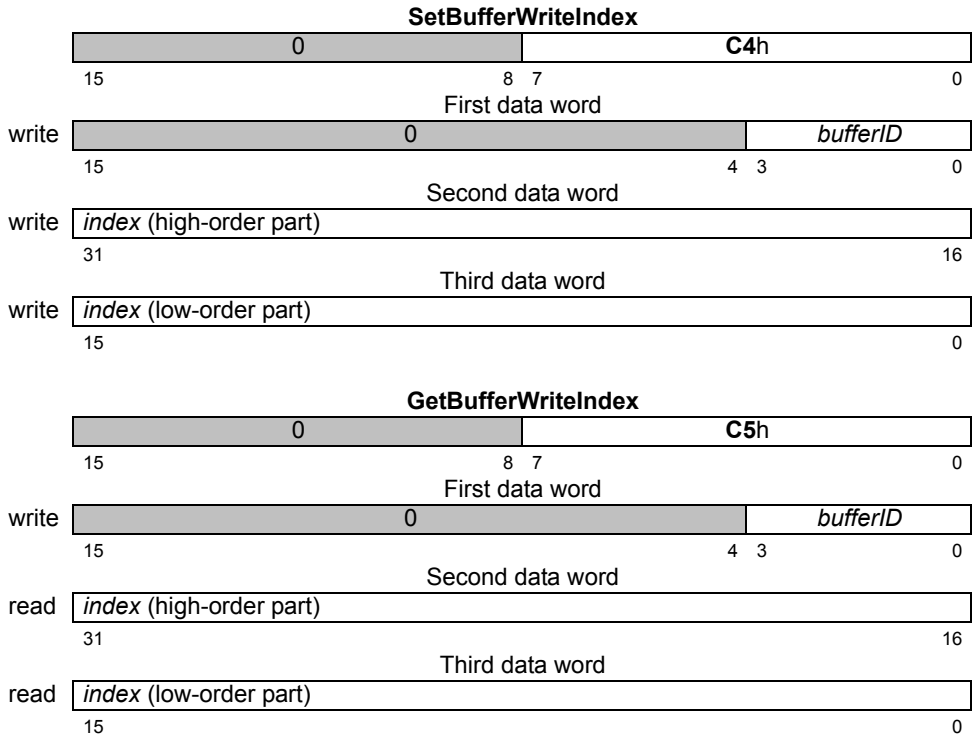
Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Type	Range	Scaling	Units
bufferID	unsigned 16 bits	0 *to* 31	unity	-
index	unsigned 32 bits	0 *to* buffer length - 1	unity	double words

Packet Structure



Description

SetBufferWriteIndex sets the write *index* for the specified *bufferID*.

GetBufferWriteIndex returns the write *index* for the specified *bufferID*.

Restrictions

C-Motion API

```
PMDresult PMDSetBufferWriteIndex(PMDAxisInterface axis_intf,
                                 PMDuint16 bufferID, PMDuint32 index);
PMDresult PMDGetBufferWriteIndex(PMDAxisInterface axis_intf,
                                 PMDuint16 bufferID, PMDuint32* index)
```

VB-Motion API

```
Dim index as Long
MagellanObject.BufferWriteIndex( bufferID ) = index
index = MagellanObject.BufferWriteIndex( bufferID )
```

see

Set/GetBufferLength (p. 81), **Set/GetBufferReadIndex** (p. 82), **Set/GetBufferStart** (p. 83)

Syntax

SetBusVoltageLimits *axis parameter value*
GetBusVoltageLimits *axis parameter*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

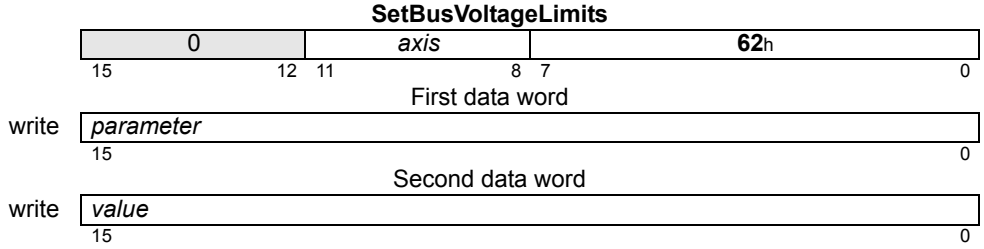
Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
parameter	*Overvoltage Limit*	0
	Undervoltage Limit	1

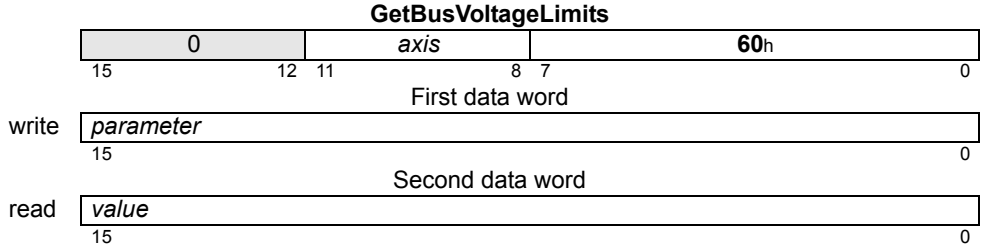
	Type	Range	Scaling
value	unsigned 16 bits	0 *to* 2^16–1	1.3612 mv/count

Packet Structure

SetBusVoltageLimits



GetBusVoltageLimits



Description

SetBusVoltageLimits sets the thresholds for determination of overvoltage and undervoltage conditions. If the bus voltage exceeds the ***Overvoltage Limit*** value, an overvoltage condition occurs. If the bus voltage is less than the ***Undervoltage Limit*** value, an undervoltage condition occurs. Both the ***Overvoltage Limit*** and ***Undervoltage Limit*** have ranges of 0 to 2^16–1, with scaling of 1.3612 millivolts/count.

For example, to set the overvoltage threshold to 30V, ***Overvoltage Limit*** should be set to 30V/1.3612mv = 22039.

GetBusVoltageLimits reads the indicated limit.

Restrictions

Get/SetBusVoltageLimits is only available in products equipped with Bus voltage sensors.

The ***Overvoltage Limit*** cannot be set to a value greater than the reset default setting, and the ***Undervoltage Limit*** cannot be set to a value less than the reset default setting.

SetBusVoltageLimits (cont.) 62h
GetBusVoltageLimits 60h

C-Motion API

```
PMDresult PMDSetBusVoltageLimits(PMDAxisInterface axis_intf,
                                 PMDuint16 parameter,
                                 PMDuint16 value)
PMDresult PMDGetBusVoltageLimits(PMDAxisInterface axis_intf,
                                 PMDuint16 parameter,
                                 PMDuint16* value)
```

VB-Motion API

```
Dim value as Short
MagellanAxis.BusVoltageLimits( parameter ) = value
value = MagellanAxis.BusVoltageLimits( parameter )
```

see **Set/GetFaultOutMask** (p. 104), **GetBusVoltage** (p. 29), **GetDriveFaultStatus** (p. 37), **ClearDriveFaultStatus** (p. 21), **GetEventStatus** (p. 39), **ResetEventStatus** (p. 64)

Syntax

SetCANMode *mode*
GetCANMode

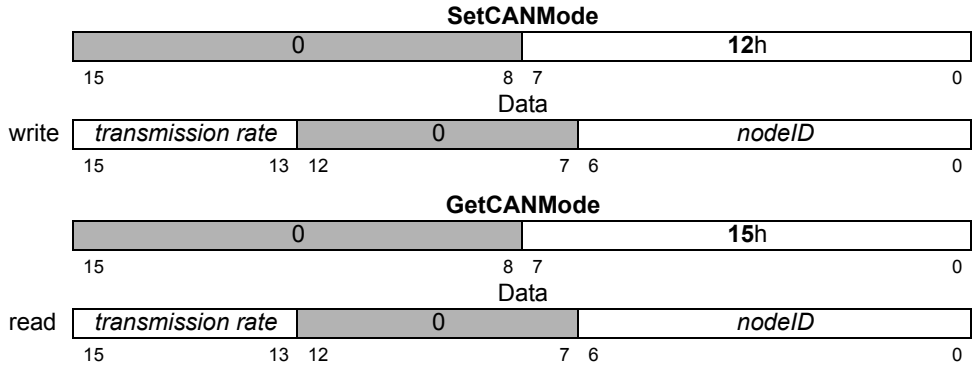
Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Type	Encoding
mode	unsigned 16 bits	see below

Packet Structure



Description

SetCANMode sets the CAN 2.0B communication parameters for the motion processor. After completion of this command, the motion processor will respond to a CAN receive message addressed to 600h + nodeID. CAN responses are sent to 580h + nodeID. The CAN transmission rate will be as specified in the *transmission rate* parameter. Note that when this command is used to change to a new nodeID, the command response (for this command) will be sent to the new nodeID. The following table shows the encoding of the data used by this command.

Bits	Name	Instance	Encoding
0–6	CAN NodeID	*Address 0*	0
		Address 1	1
		...	...
		Address 127	127
7–12	— (Reserved)		
13–15	Transmission Rate	*1,000,000 baud*	0
		800,000	1
		500,000	2
		250,000	3
		125,000	4
		50,000	5
		20,000	6
		10,000	7

Restrictions

C-Motion API

```
PMDresult PMDSetCANMode(PMDAxisHandle axis_handle, PMDuint8 nodeID,
                        PMDuint8 transmission_rate)
PMDresult PMDGetCANMode(PMDAxisHandle axis_handle, PMDuint8* nodeID,
                        PMDuint8* transmission_rate)
```

VB-Motion API

```
CommunicationCAN.CANModeSet( [ in ] nodeID, [ in ] transmission_rate )
```

see

SetCaptureSource D8h
GetCaptureSource D9h

Syntax

SetCaptureSource *axis source*
GetCaptureSource *axis*

Motor Types

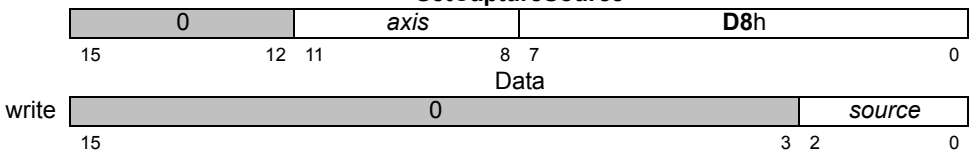
DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

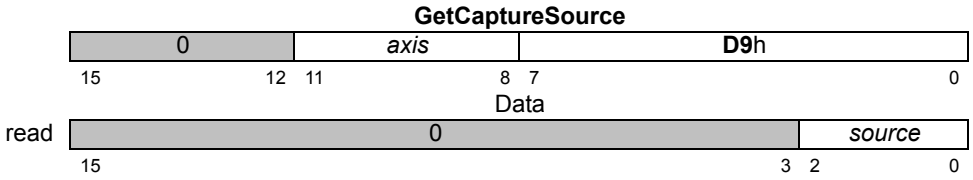
Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
source	*Index*	0
	Home	1
	High Speed Capture	2

Packet Structure

SetCaptureSource



GetCaptureSource



Description

SetCaptureSource determines which of three signals—*Index*, *Home*, or *High Speed Capture*—is used to trigger the capture of the actual axis position for the specified ***axis***. **GetCaptureSource** returns the capture signal ***source*** for the selected ***axis***.

Restrictions

High Speed Capture is not available as a capture source in all products. See the product user's guide.

C-Motion API

```
PMDresult PMDSetCaptureSource(PMDAxisInterface axis_intf,
                              PMDuint16 source)
PMDresult PMDGetCaptureSource(PMDAxisInterface axis_intf,
                              PMDuint16* source)
```

VB-Motion API

```
Dim source as Short
MagellanAxis.CaptureSource = source
source = MagellanAxis.CaptureSource
```

see

GetCaptureValue (p. 30)

Syntax

SetCommutationMode *axis mode*
GetCommutationMode *axis*

Motor Types

	Brushless DC		

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
mode	*Sinusoidal*	0
	Hall-based	1

Packet Structure

SetCommutationMode

	0	*axis*	**E2**h
	15 12	11 8	7 0
	Data		
write	0		*mode*
	31 1		0

GetCommutationMode

	0	*axis*	**E3**h
	15 12	11 8	7 0
	Data		
read	0		*mode*
	31 1		0

Description

SetCommutationMode sets the phase commutation *mode* for the specified *axis*.

When set to *Sinusoidal*, as the motor turns, the encoder input signals are used to calculate the phase angle. This angle is in turn used to generate sinusoidally varying outputs to each motor winding.

When set to *Hall-based,* the Hall effect sensor inputs are used to commutate the motor windings using a "six-step" or "trapezoidal" waveform method.

When using FOC current control, this command is used to define the method used for motor phase determination.

GetCommutationMode returns the value of the commutation mode.

Restrictions

C-Motion API

```
PMDresult PMDSetCommutationMode(PMDAxisInterface axis_intf,
                                PMDuint16 mode)
PMDresult PMDGetCommutationMode(PMDAxisInterface axis_intf,
                                PMDuint16* mode)
```

VB-Motion API

```
Dim mode as Short
MagellanAxis.CommutationMode = mode
mode = MagellanAxis.CommutationMode
```

see

Set/GetPhasePrescale (p. 132), **Set/GetPhaseCounts** (p. 128)

SetCurrentControlMode buffered 43h
GetCurrentControlMode 44h

Syntax

SetCurrentControlMode *axis mode*
GetCurrentControlMode *axis*

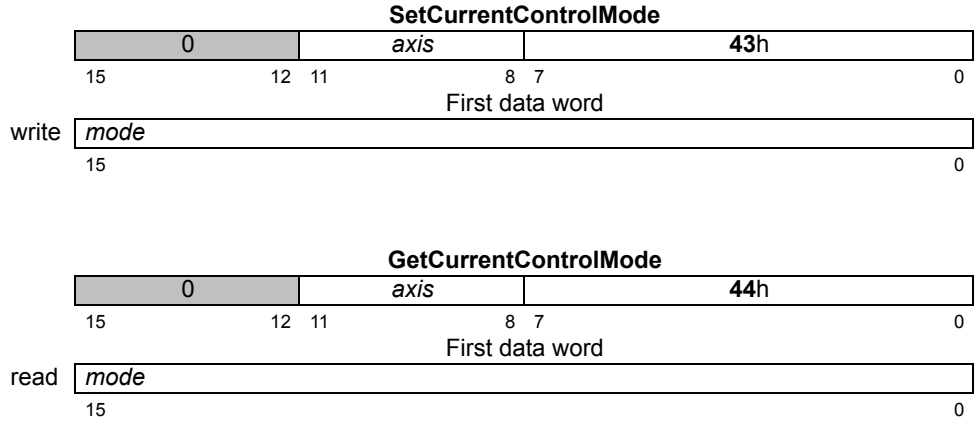
Motor Types

	Brushless DC	Microstepping	

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
mode	*Phase A /B Current Loops*	0
	FOC	1

Packet Structure



Description

SetCurrentControlMode configures the *axis* to use either the *Phase A/B* method or the *FOC* method for current control.

GetCurrentControlMode gets the buffered current loop *mode* for the indicated axis.

Restrictions

This command is only available on products that include a digital current loop.

SetCurrentControlMode is a buffered command. It will not take effect until an update is done on the current loop (through **Update** command, **MultiUpdate** command, or update action on breakpoint). The value read by **GetCurrentControlMode** is the buffered setting.

C-Motion API

```
PMDresult PMDSetCurrentControlMode(PMDAxisInterface axis_intf,
                                   PMDuint16 mode)
PMDresult PMDGetCurrentControlMode(PMDAxisInterface axis_intf,
                                   PMDuint16* mode)
```

VB-Motion API

```
Dim mode as Short
MagellanAxis.CurrentControlMode = mode
mode = MagellanAxis.CurrentControlMode
```

see

Update (p. 167), **Set/GetUpdateMask** (p. 163), **MultiUpdate** (p. 55), **Set/GetBreakpointUpdateMask** (p. 77), **GetFOCValue** (p. 40), **Get/SetFOC** (p. 106), **GetCurrentLoopValue** (p. 35), **Get/SetCurrentLoop** (p. 93)

Syntax

SetCurrentFoldback *axis parameter value*
GetCurrentFoldback *axis parameter*

Motor Types

DC Brush	Brushless DC	Microstepping	

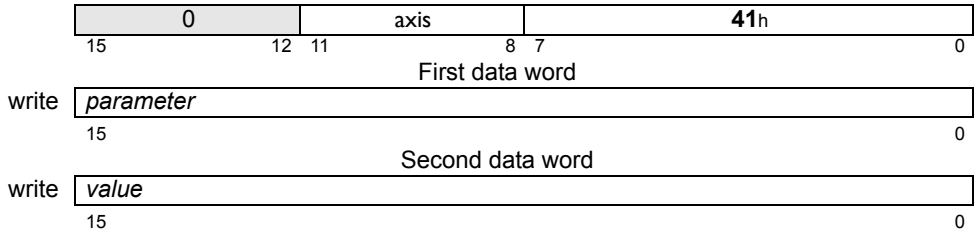
Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
parameter	*Continuous Current Limit*	0
	Energy Limit	1

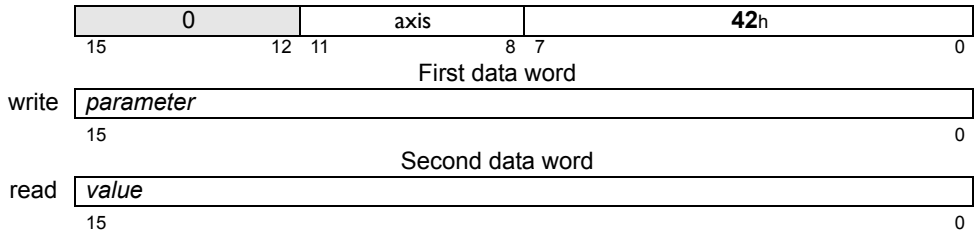
	Type	Range/Scaling
value	unsigned 16-bit	see below

Packet Structure

SetCurrentFoldback



GetCurrentFoldback



Description

SetCurrentFoldback is used to set various I^2T foldback-related parameters. Two parameters can be set, the ***Continuous Current Limit***, and the ***Energy Limit***. The units of ***Continuous Current Limit*** are convertible to milliAmps, and represent percentage of maximum peak current, with scaling of 100/2^15. The range is from 0% to the factory default continuous current limit setting. When using this command with the ION drive, check the *ION Digital Drive User's Manual* for exact scaling values. Different drives have different scaling values and default limit settings.

The units of ***Energy Limit*** are convertible to $Amp^2Seconds$, and represent the percentage of maximum energy, with scaling of 100/2^15. The range is from 0% to the factory default energy limit setting. When using this command with the ION drive, check the *ION Digital Drive User's Manual* for exact scaling values. Different drives have different scaling values and default limit settings.

The ***Continuous Current Limit*** is used by the current foldback algorithm. When the current output of the drive exceeds this setting, accumulation of the I^2 energy above this setting begins. Once the accumulated excess I^2 energy exceeds the value specified by the ***Energy Limit*** parameter, a current foldback condition

Description (cont.)

exists and the commanded current will be limited to the specified *Continuous Current Limit*. When this occurs, the Current Foldback bit in the Event Status and Drive Status registers will be set. When the accumulated I^2 energy above the *Continuous Current Limit* drops to zero (0), the limit is removed, and the Current Foldback bit in the Drive Status register is cleared.

SetEventAction can be used to configure a change in operating mode when current foldback occurs. Doing this does not interfere with the basic operation of Current Foldback described above. If this is done, the Current Foldback bit in the Event Status register must be cleared prior to restoring the operating mode, regardless of whether the system is in current foldback or not.

When current control is not active, a current foldback event always causes a change to the disabled state (all loops and motor output are disabled), regardless of the programmed Event Action. Changing the operating mode from disabled requires clearing of the Current Foldback bit in Event Status.

GetCurrentFoldback gets the maximum continuous current setting.

Restrictions

This command is only available on products that support digital current control.

Values of *Continuous Current Limit* greater than the factory setting for maximum continuous current are not allowed.

C-Motion API

```
PMDresult PMDSetCurrentFoldback(PMDAxisInterface axis_intf,
                                PMDuint16 parameter,
                                PMDuint16 value)
PMDresult PMDGetCurrentFoldback(PMDAxisInterface axis_intf,
                                PMDuint16 parameter,
                                PMDuint16* value)
```

VB-Motion API

```
Dim value as Short
MagellanAxis.CurrentFoldback( parameter ) = value
value = MagellanAxis.CurrentFoldback( parameter )
```

see

GetEventStatus (p. 39), **ResetEventStatus** (p. 64), **GetDriveStatus** (p. 38), **RestoreOperatingMode** (p. 65), **GetActiveOperatingMode** (p. 25)

Syntax

SetCurrentLoop *axis phase parameter value*
GetCurrentLoop *axis phase parameter*

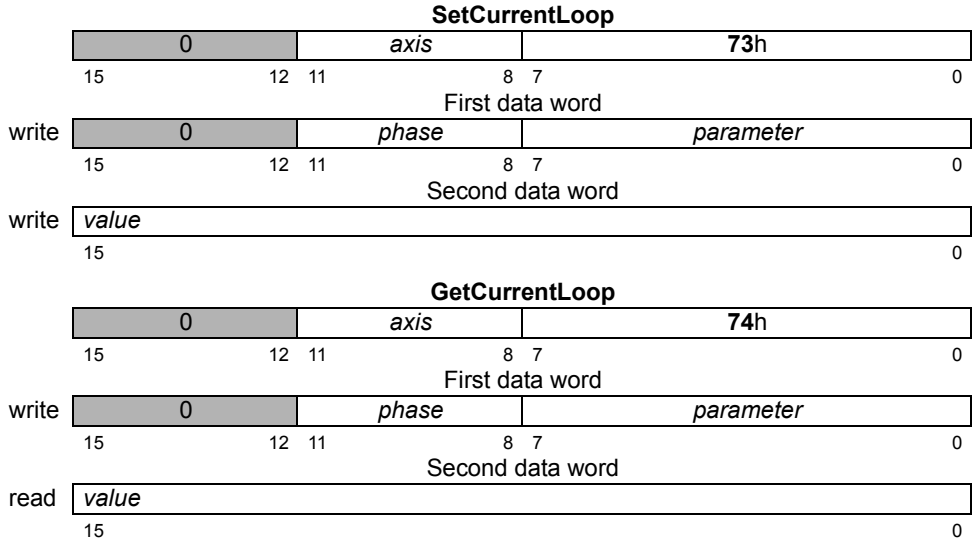
Motor Types

DC Brush	Brushless DC	Microstepping	

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
phase	*Phase A*	0
	Phase B	1
	Both (A and B)	2
parameter	*Proportional Gain (KpCurrent)*	0
	Integral Gain (KiCurrent)	1
	Integral Sum Limit (ILimitCurrent)	2
	Type	**Range/Scaling**
value	unsigned 16 bits	see below

Packet Structure



Description

Set/GetCurrentLoop is used to configure the operating parameters of the Phase A/B PI digital current loops. See the product user's guide for more information on how each ***parameter*** is used in the current loop processing. The ***value*** written/read is always an unsigned 16-bit value, with the parameter-specific scaling shown below:

Parameter	Range	Scaling	Units
Proportional Gain (KpCurrent)	0 to 2^15–1	1/64	gain
Integral Gain (KiCurrent)	0 to 2^15–1	1/256	gain/cycles
Integral Sum Limit (ILimitCurrent)	0 to 2^15–1	1/100	% current * cycles

A setting of 64 for ***KpCurrent*** corresponds to a gain of 1. That is, an error signal of 30% maximum current will cause the proportional contribution of the current loop output to be 30% of maximum output. Similarly, setting ***KiCurrent*** to 256 gives it a gain of 1, and the value of the integral sum would become the integral contribution to the output. The units of time for the integral sum are cycles.

Description (cont.)

ILimitCurrent is used to limit the contribution of the integral sum at the output. Its effect depends on the value of *KiCurrent*. Setting *ILimitCurrent* to 1000 when *KiCurrent* is 10 means that the maximum contribution to the output is 1000 x 10 = 10,000 out of 2^15 - 1 or approximately 30.5%

The *phase* argument can be used to set the operating parameters for the A and B loops independently. In most cases, the A and B loops will not require different operating parameters, so **SetCurrentLoop** can be used with a *phase* of 2, which sets both the A and B loops in a single API command. For **GetCurrentLoop**, a *phase* of 2 is not valid.

Restrictions

Set/GetCurrentLoop are buffered commands. All parameters set are buffered, and will not take effect until an update is done on the current loop (through **Update** command, **MultiUpdate** command, or update action on breakpoint). The values read by **GetCurrentLoop** are the buffered settings.

This command is only supported in products that include digital current control, and when the current control mode is Phase A/B.

C-Motion API

```
PMDresult PMDSetCurrentLoop(PMDAxisInterface axis_intf,
                            PMDuint8 phase,
                            PMDuint8 parameter,
                            PMDuint16 value)
PMDresult PMDGetCurrentLoop(PMDAxisInterface axis_intf,
                            PMDuint8 phase,
                            PMDuint8 parameter,
                            PMDuint16* value)
```

VB-Motion API

```
MagellanAxis.CurrentLoopSet ( [in] phase,
                              [in] parameter,
                              [in] value )
MagellanAxis.CurrentLoopGet ( [in] phase,
                              [in] parameter,
                              [out] value )
```

see

Update (p. 167), **Set/GetUpdateMask** (p. 163), **MultiUpdate** (p. 55), **Set/GetBreakpointUpdateMask** (p. 77), **GetCurrentLoopValue** (p. 35), **Set/GetCurrentControlMode** (p. 90)

Syntax

SetDeceleration *axis deceleration*
GetDeceleration *axis*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

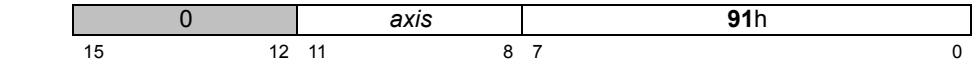
Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

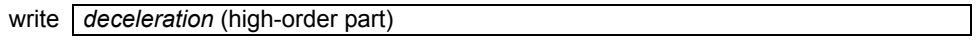
	Type	Range	Scaling	Units
deceleration	unsigned 32 bits	0 *to* $2^{31}-1$	$1/2^{16}$	counts/cycle^2 microsteps/cycle^2

Packet Structure

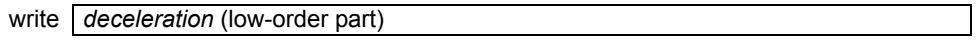
SetDeceleration



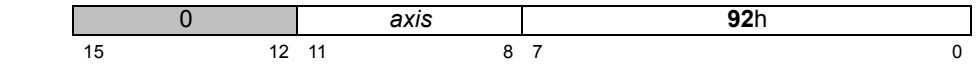
First data word



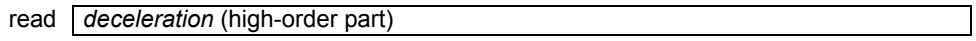
Second data word



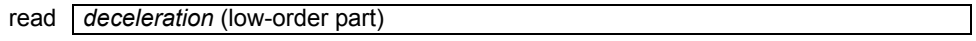
GetDeceleration



First data word



Second data word



Description

SetDeceleration loads the maximum deceleration buffer register for the specified *axis*.

GetDeceleration returns the value of the maximum deceleration buffer.

Scaling example: To load a value of 1.750 counts/cycle^2 multiply by 65,536 (giving 114,688) and load the resultant number as a 32-bit number, giving 0001 in the high word and C000h in the low word. Retrieved numbers (**GetDeceleration**) must correspondingly be divided by 65,536 to convert to units of counts/cycle^2 or steps/cycle^2

Restrictions

This is a buffered command. The new value set will not take effect until the next **Update** or **MultiUpdate** command is entered, with the Trajectory Update bit set in the update mask.

These commands are used with the Trapezoidal and Velocity Contouring profile modes. They are not used with the Electronic Gear or S-curve profile mode.

Note: If *deceleration* is set to zero (0), then the value specified for acceleration (**SetAcceleration**) will automatically be used to set the magnitude of deceleration.

C-Motion API

```
PMDresult PMDSetDeceleration(PMDAxisInterface axis_intf,
                             PMDuint32 deceleration)
PMDresult PMDGetDeceleration(PMDAxisInterface axis_intf,
                             PMDuint32* deceleration)
```

VB-Motion API

```
Dim deceleration as Long
MagellanAxis.Deceleration = deceleration
deceleration = MagellanAxis.Deceleration
```

see

Set/GetAcceleration (p. 66), **Set/GetPosition** (p. 133), **Set/GetVelocity** (p. 165), **MultiUpdate** (p. 55), **Update** (p. 167)

SetDefault 89h
GetDefault 8Ah

Syntax

SetDefault *axis variable value*
GetDefault *axis variable*

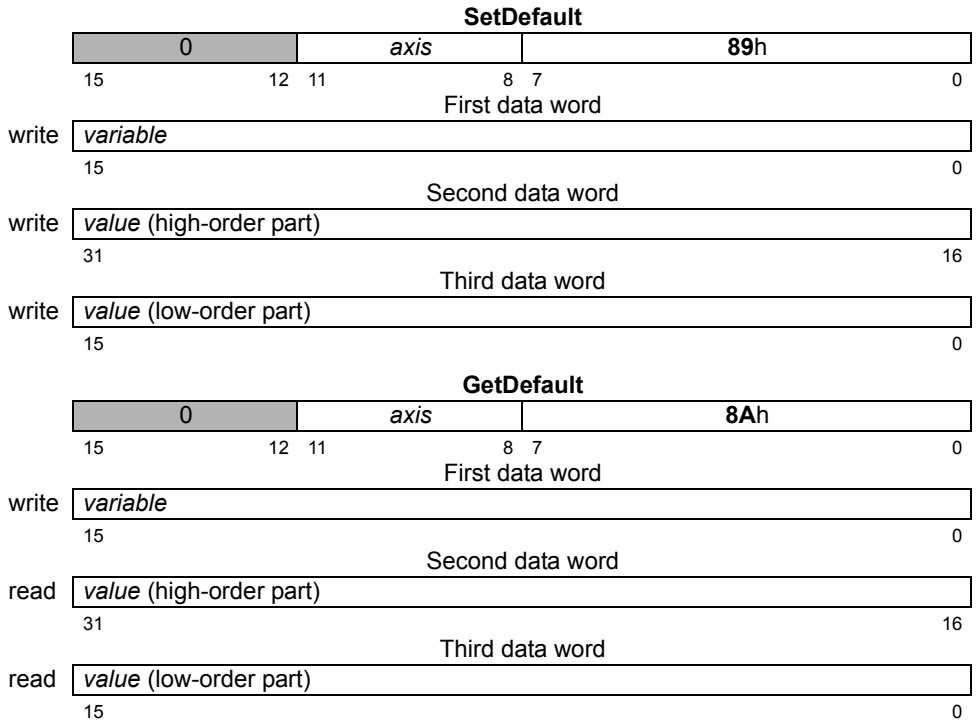
Motor Type

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
variable	*CanMode*	0
	SerialPortMode485	1
	Type	**Range/Scaling**
value	32 bits	see below

Packet Structure



Description

SetDefault is used to override the reset default settings of system variables. When **SetDefault** is invoked to change the reset default of a ***variable***, it stores the ***value*** sent by the user in non-volatile memory. It does not modify the value of the variable in active use. On subsequent system power cycles or resets, this ***value*** will become the default for the selected ***variable***.

The value for each variable is the value that would be used normally by the "Set/Get" command for that variable. When configuring variables that are 16-bit values, the value should be sent as the low order part of the 32-bit ***value***.

The ***axis*** sent with **Set/GetDefault** may or may not be relevent, depending on whether the parameter is an axis-specific parameter or not.

GetDefault gets the reset default value of the indicated ***variable*** from non-volatile memory.

Restrictions

This command is only available in products with non-volatile memory.

The **SetDefault** command can only be executed when motor output is disabled (e.g., immediately after power-up or reset).

C-Motion API

```
PMDresult PMDSetDefault (PMDAxisInterface axis_intf,
                         PMDuint16 variable,
                         PMDuint32 value)
PMDresult PMDGetDefault (PMDAxisInterface axis_intf,
                         PMDuint16 variable,
                         PMDuint32* value)
```

VB-Motion API

```
MagellanAxis.DefaultSet( [in] variable, [in] value )
MagellanAxis.DefaultGet( [in] variable, [out] value )
```

see

Reset (p. 60)

SetEncoderModulus 8Dh
GetEncoderModulus 8Eh

Syntax

SetEncoderModulus *axis modulus*
GetEncoderModulus *axis*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

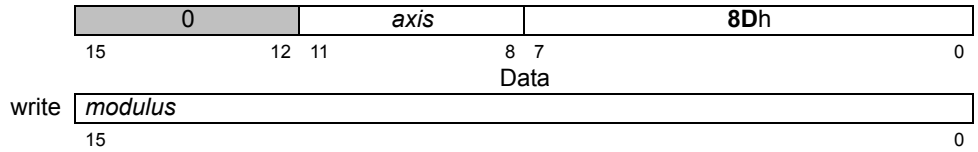
Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

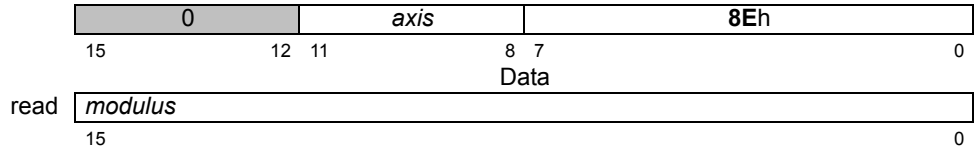
	Type	Range	Scaling	Units
modulus	unsigned 16 bits	0 *to* 2^{15}–1	unity	counts

Packet Structure

SetEncoderModulus



GetEncoderModulus



Description

SetEncoderModulus sets the parallel word range for the specified ***axis*** when parallel-word feedback is used. The ***modulus*** determines the range of the connected device. For multi-turn systems, this value is used to determine when a position wrap condition has occurred. The value provided should be one half of the actual range of the axis. For example, if the parallel-word input is used with a linear potentiometer connected to an external A/D (Analog to Digital converter) which has 12 bits of resolution, then the total range is 4,096 and a value of 2,048 should be loaded with this command.

GetEncoderModulus returns the encoder modulus.

Restrictions

A value for encoder modulus is only required when the encoder source is set to parallel.

C-Motion API

```
PMDresult PMDSetEncoderModulus(PMDAxisInterface axis_intf,
                               PMDuint16 modulus)
PMDresult PMDGetEncoderModulus(PMDAxisInterface axis_intf,
                               PMDuint16* modulus)
```

VB-Motion API

```
Dim modulus as Short
MagellanAxis.EncoderModulus = modulus
modulus = MagellanAxis.EncoderModulus
```

see

Set/GetEncoderSource (p. 99)

Syntax

SetEncoderSource *axis source*
GetEncoderSource *axis*

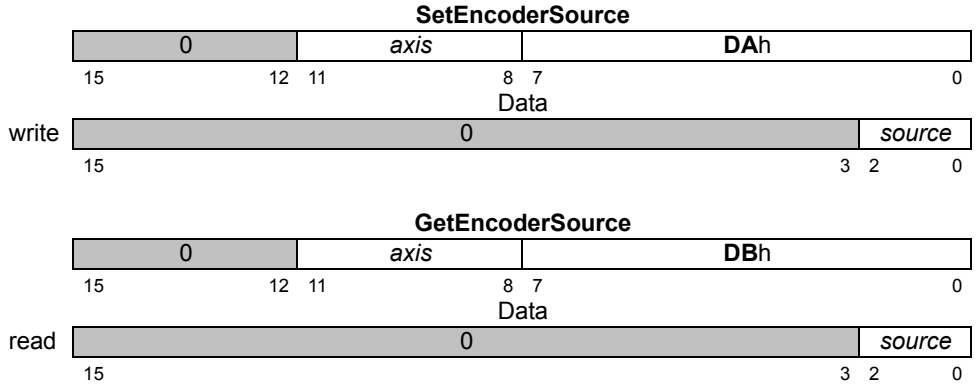
Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
source	*Incremental*	0
	Parallel	1
	None	2
	Loopback	3
	Pulse and Direction	4

Packet Structure



Description

SetEncoderSource sets the type of encoder feedback (*Incremental* quadrature encoder or *Parallel*-word) for the specified *axis*. When incremental quadrature is selected the motion processor expects A and B quadrature signals to be input at the QuadA and QuadB axis inputs. When parallel-word is selected the motion processor expects user-defined external circuitry connected to the motion processor's external bus to load a 16-bit word containing the current position value for the selected axis. External feedback devices with less than 16 bits may be used but the unused bits must be sign extended or zeroed.

When motor type (see **SetMotorType** (p. 120)) is set to *Pulse and Direction* and the encoder source is set to *Loopback*, the step output is internally fed back into the quadrature counters. This allows for position capture of the step position when a physical encoder is not present.

When the encoder source is set to *Pulse and Direction,* then Magellan expects the incoming position encoding to correspond to a pulse & direction encoding scheme rather than a quadrature encoding scheme. This feature is most commonly used with electronic gear mode, so that the Magellan processor can be driven by a motion controller that outputs pulse & direction signals.

GetEncoderSource returns the code for the current type of feedback.

Restrictions

A *Loopback* source is only supported for pulse & direction motors. *Loopback* is not supported in single-chip versions (MC58110 & MC55110). A source value of *None* is typically only used with microstepping and pulse & direction motors.

Not all products support all types of encoders. See the product user's guide.

SetEncoderSource (cont.) DAh
GetEncoderSource DBh

C-Motion API

```
PMDresult PMDSetEncoderSource(PMDAxisInterface axis_intf, PMDuint16 source)
PMDresult PMDGetEncoderSource(PMDAxisInterface axis_intf, PMDuint16* source)
```

VB-Motion API

```
Dim source as Short
MagellanAxis.EncoderSource = source
source = MagellanAxis.EncoderSource
```

see **Set/GetEncoderModulus** (p. 98)

SetEncoderToStepRatio DEh
GetEncoderToStepRatio DFh

Syntax

SetEncoderToStepRatio *axis counts steps*
GetEncoderToStepRatio *axis*

Motor Types

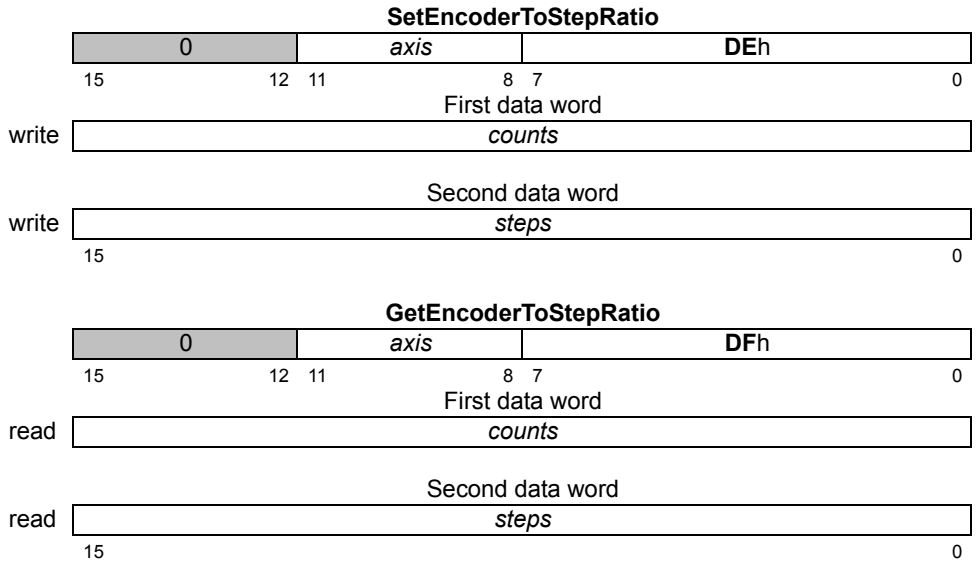
		Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

	Type	Range	Scaling	Units
counts	unsigned 16 bits	1 *to* 2^{15}–1	unity	counts
steps	unsigned 16 bits	1 *to* 2^{15}–1	unity	microsteps

Packet Structure



Description

SetEncoderToStepRatio sets the ratio of the number of encoder counts to the number of output steps per motor rotation used by the motion processor to convert encoder counts into steps. ***Counts*** is the number of encoder counts per full rotation of the motor. ***Steps*** is the number of steps output by the motion processor per full rotation of the motor. Since this command sets a ratio, the parameters do not have to be for a full rotation as long as they correctly represent the encoder count to step ratio. **GetEncoderToStepRatio** returns the ratio of the number of encoder counts to the number of output steps per motor rotation.

C-Motion API

```
PMDresult PMDSetEncoderToStepRatio(PMDAxisInterface axis_intf,
                                   PMDuint16 counts, PMDuint16 steps)
PMDresult PMDGetEncoderToStepRatio(PMDAxisInterface axis_intf,
                                   PMDuint16* counts, PMDuint16* steps)
```

VB-Motion API

```
MagellanAxis.EncoderToStepRatioSet( [in] counts, [in] steps )
MagellanAxis.EncoderToStepRatioGet( [out] counts, [out] steps )
```

see

Set/GetActualPositionUnits (p. 70)

SetEventAction 48h
GetEventAction 49h

Syntax

SetEventAction *axis event action*
GetEventAction *axis event*

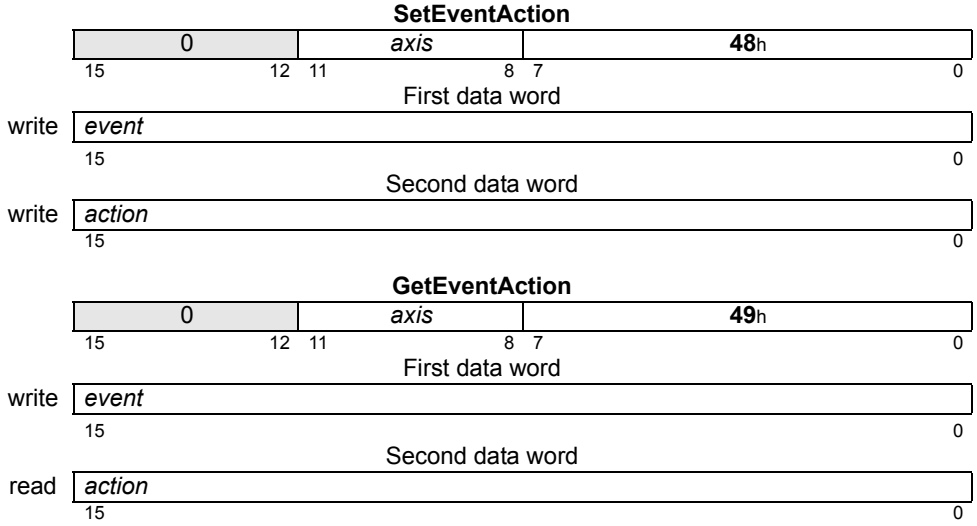
Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
event	*Immediate*	0
	Positive Limit	1
	Negative Limit	2
	Motion Error	3
	Current Foldback	4
action	*None*	0
	— (Reserved)	1
	Abrupt Stop	2
	Smooth Stop	3
	— (Reserved)	4
	Disable Position Loop & Higher Modules	5
	Disable Current Loop & Higher Modules	6
	Disable Motor Output & Higher Modules	7
	Abrupt Stop with Position Error Clear	8

Packet Structure



Description

SetEventAction configures what actions will be taken by the ***axis*** in response to a given ***event***. The ***action*** can be either to modify the operating mode by disabling some or all of the loops, or, in the case of all loops remaining on, to perform an abrupt or smooth stop. The ***Abrupt Stop*** action can be done with or without a clearing of the position error.

Description (cont.)

When, through **SetEventAction**, one of the *events* causes an *action*, the event bit in the Event Status register must be cleared prior to returning to operation. For trajectory stops, this means that the bit must be cleared prior to performing another trajectory move. For changes in operating mode, this means that the bit must be cleared prior to restoring the operating mode, either by **RestoreOperatingMode** or **SetOperatingMode**.

An exception is the Motion Error event, which only needs to be cleared in Event Status if its *action* is *Abrubt Stop* or *Smooth Stop*. If it causes changes in operating mode, the operating mode can be restored without clearing the bit in Event Status first.

GetEventAction gets the action that is currently programmed for the given event with the exception of the *Immediate* event, which cannot be read back.

Restrictions

If a *Smooth Stop* action occurs while the trajectory mode is S-curve, the trajectory cannot be restarted until the smooth stop is complete. If a *Smooth Stop* action occurs while the trajectory mode is electronic gearing, an abrupt stop will occur.

C-Motion API

```
PMDresult PMDSetEventAction (PMDAxisInterface axis_intf,
                             PMDuint16 event,
                             PMDuint16 action)
PMDresult PMDGetEventAction (PMDAxisInterface axis_intf,
                             PMDuint16 event,
                             PMDuint16* action)
```

VB-Motion API

```
Dim action as Short
MagellanAxis.EventAction( event ) = action
action = MagellanAxis.EventAction( event )
```

see

GetActiveOperatingMode (p. 25), **RestoreOperatingMode** (p. 65), **Set/GetOperatingMode** (p. 122)

SetFaultOutMask FBh
GetFaultOutMask FCh

Syntax

SetFaultOutMask *axis mask*
GetFaultOutMask *axis*

Motor Types

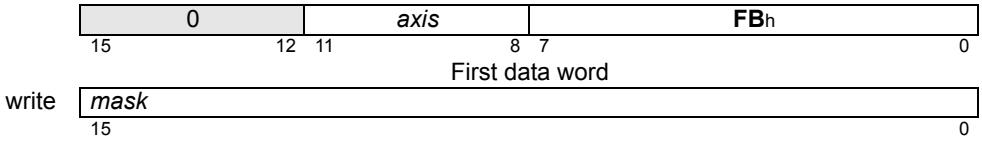
DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

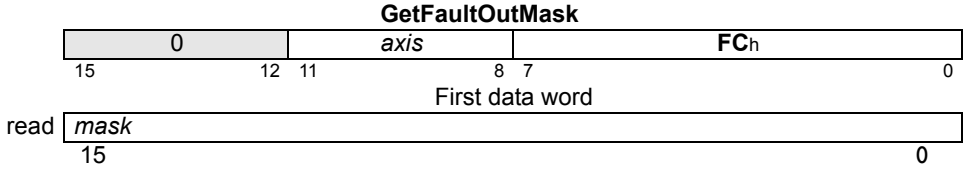
Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
mask	see below	bitmask

Packet Structure

SetFaultOutMask



GetFaultOutMask



Description

SetFaultOutMask configures the mask on Event Status register bits that will be ORed together on the FaultOut pin. The FaultOut pin is active high, as are the bits in Event Status. Thus, FaultOut will go high when any of the enabled bits in Event Status are set (1). The ***mask*** parameter is used to determine what bits in the Event Status register can cause FaultOut high, as follows:

Name	Bit
Motion Complete	0
Wrap-around	1
Breakpoint 1	2
Position Capture	3
Motion Error	4
Positive Limit	5
Negative Limit	6
Instruction Error	7
Disable	8
Overtemperature Fault	9
Bus Voltage Fault	10
Commutation Error	11
Current Foldback	12
— (Reserved)	13
Breakpoint 2	14
— (Reserved)	15

Description (cont.)

For example, a *mask* setting of hexadecimal 0610h will configure the FaultOut pin to go high upon a motion error, Overtemperature Fault, or Bus Voltage Fault. The FaultOut pin stays high until all Fault enabled bits in Event Status are cleared. The default value for the FaultOut *mask* is 0600h – Overtemperature Fault and Bus Voltage Fault enabled.

GetFaultOutMask gets the current *mask* for the indicated *axis*.

Restrictions

This command is only available on products that include a FaultOut pin.

Depending on the product, all of the specified bits in Event Status may not be available.

In addition to the FaultOut *mask* on the Event Status register, the FaultOut pin is driven by a mask on the Drive Fault Status register (bits 4, 2, 1, and 0) which cannot be changed, and is internally ORed with the FaultOut *mask* on Event Status.

C-Motion API

```
PMDresult PMDSetFaultOutMask (PMDAxisInterface axis_intf,
                              PMDuint16 mask)
PMDresult PMDGetFaultOutMask (PMDAxisInterface axis_intf,
                              PMDuint16* mask)
```

VB-Motion API

```
Dim mask as Short
MagellanAxis.FaultOutMask = mask
mask = MagellanAxis.FaultOutMask
```

see

Set/GetInterruptMask (p. 113)

Syntax

SetFOC *axis loop parameter value*
GetFOC *axis loop parameter*

Motor Types

	Brushless DC	Microstepping	

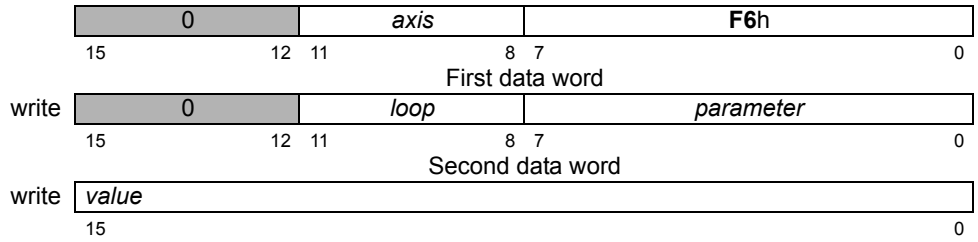
Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
loop	*Direct(D)*	0
	Quadrature(Q)	1
	Both(D and Q)	2
parameter	*Proportional Gain (KpDQ)*	0
	Integral Gain (KiDQ)	1
	Integral Sum Limit (ILimitDQ)	2

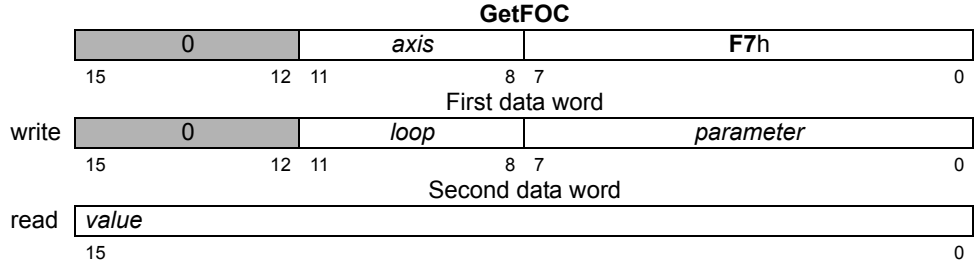
	Type	Range/Scaling
value	unsigned 16 bits	see below

Packet Structure

SetFOC



GetFOC



Description

Set/GetFOC is used to configure the operating parameters of the FOC-Current control. See the product user's guide for more information on how each ***parameter*** is used in the current loop processing. The ***value*** written/read is always an unsigned 16-bit value, with the parameter-specific scaling shown below:

Parameter	Range	Scaling	Units
Proportional Gain (KpDQ)	0 to 2^15–1	1/64	gain
Integral Gain (KiDQ)	0 to 2^15–1	1/256	gain/cycles
Integral Sum Limit (ILimitDQ)	0 to 2^15–1	1/100	% current * cycles

A setting of 64 for ***KpDQ*** corresponds to a gain of 1. That is, an error signal of 30% maximum current will cause the proportional contribution of the current loop output to be 30% of maximum output.

Description (cont.)

Similarly, setting *KiDQ* to 256 gives it a gain of 1; the value of the integral sum would become the integral contribution to the output.

ILimitDQ is used to limit the contribution of the integral sum at the output. Its effect depends on the value of *KiDQ*. Setting *IlimitDQ* to 1000 when *KiDQ* is 10 means that the maximum contribution to the output is 1000 x 10 = 10,000 out of 2^15 - 1 or approximately 30.5%. The units of time for the integral sum are cycles.

The *loop* argument allows individual configuration of the parameters for the D and Q current loops. Alternately, a *loop* of 2 can be used with **SetFOC** to set the D and Q loops with a single API command. A *loop* of 2 is not valid for **GetFOC**.

Restrictions

Set/GetFOC are buffered commands. All parameters set are buffered, and will not take effect until an update is done on the current loop (through **Update** command, **MultiUpdate** command, or update action on breakpoint). The values read by **GetFOC** are the buffered settings.

These commands are only supported in products that include digital current control, and when the current control mode is set to FOC.

C-Motion API

```
PMDresult PMDSetFOC (PMDAxisInterface axis_intf,
                     PMDuint8 loop,
                     PMDuint8 parameter,
                     PMDuint16 value)
PMDresult PMDGetFOC (PMDAxisInterface axis_intf,
                     PMDuint8 loop,
                     PMDuint8 parameter,
                     PMDuint16* value)
```

VB-Motion API

```
MagellanAxis.FOCSet( [in] loop, [in] parameter, [in] value )
MagellanAxis.FOCGet( [in] loop, [in] parameter, [out] value )
```

see

Update (p. 167), **Set/GetUpdateMask** (p. 163), **MultiUpdate** (p. 55), **Set/GetBreakpointUpdateMask** (p. 78), **GetFOCValue** (p. 40), **Set/GetCurrentControlMode** (p. 90)

Syntax

SetGearMaster *axis masterAxis source*
GetGearMaster *axis*

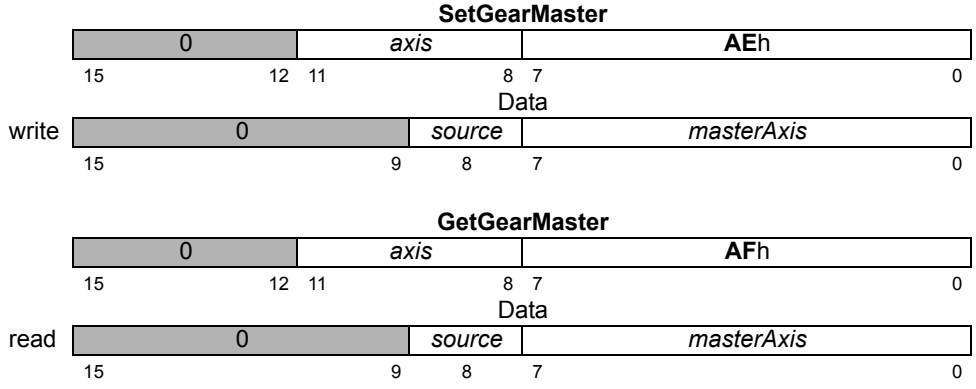
Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
masterAxis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
source	*Actual*	0
	Commanded	1

Packet Structure



Description

SetGearMaster establishes the slave (***axis***) and master (***masterAxis***) axes for the electronic-gearing profile, and sets the ***source***, ***Actual*** or ***Commanded***, of the master axis position data to be used.

The ***masterAxis*** determines the axis that will drive the slave axis. Both the slave and the master axes must be enabled (**SetOperatingMode** command). The source determines whether the master axis' commanded position as determined by the trajectory generator will be used to drive the slave axis, or whether the master axis' encoder position will be used to drive the slave.

GetGearMaster returns the value for the geared axes and position source.

Restrictions

SetGearMaster (cont.) AEh
GetGearMaster AFh

C-Motion API

```
PMDresult PMDSetGearMaster(PMDAxisInterface axis_intf,
                           PMDAxis masterAxis, PMDuint8 source)
PMDresult PMDGetGearMaster(PMDAxisInterface axis_intf,
                           PMDAxis* masterAxis, PMDuint8* source)
```

VB-Motion API

```
MagellanAxis.GearMasterSet( [in] masterAxis, [in] source )
MagellanAxis.GearMasterGet( [out] masterAxis, [out] source )
```

see **Set/GetGearRatio** (p. 110)

Syntax

SetGearRatio *slaveAxis ratio*
GetGearRatio *slaveAxis*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

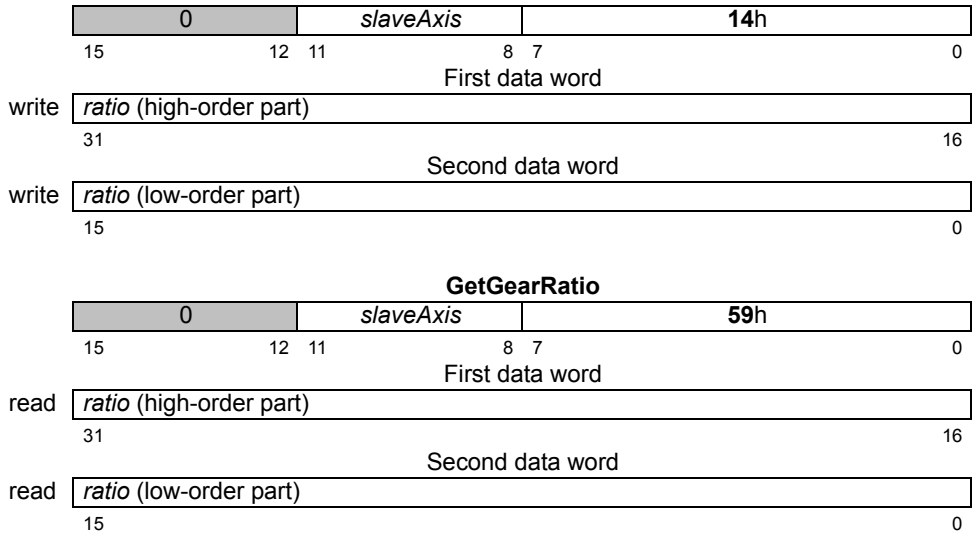
Arguments

Name	Instance	Encoding
slaveAxis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

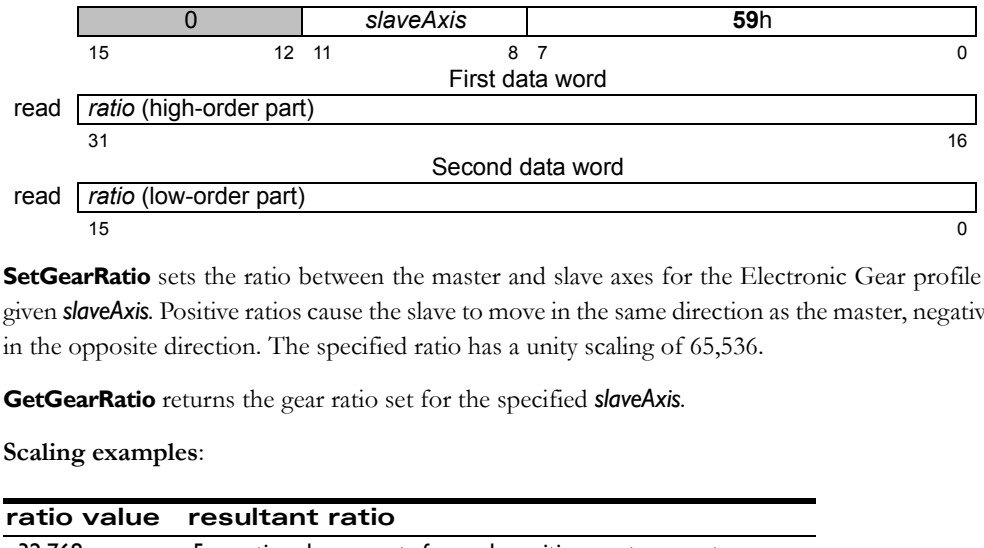
	Type	Range	Scaling	Units
ratio	signed 32 bits	-2^{31} *to* $2^{31}-1$	$1/2^{16}$	SlaveCts/MasterCts

Packet Structure

SetGearRatio



GetGearRatio



Description

SetGearRatio sets the ratio between the master and slave axes for the Electronic Gear profile for the given *slaveAxis*. Positive ratios cause the slave to move in the same direction as the master, negative ratios in the opposite direction. The specified ratio has a unity scaling of 65,536.

GetGearRatio returns the gear ratio set for the specified *slaveAxis*.

Scaling examples:

ratio value	resultant ratio
–32,768	.5 negative slave counts for each positive master count
1,000,000	15.259 positive slave counts for each positive master count
123	.0018 positive slave counts for each positive master count

Restrictions

This is a buffered command. The new value set will not take effect until the next **Update** or **MultiUpdate** command is entered, with the Trajectory Update bit set in the update mask.

C-Motion API

```
PMDresult PMDSetGearRatio(PMDAxisInterface axis_intf, PMDint32 ratio)
PMDresult PMDGetGearRatio(PMDAxisInterface axis_intf, PMDint32* ratio)
```

VB-Motion API

```
Dim ratio as Long
MagellanAxis.GearRatio = ratio
ratio = MagellanAxis.GearRatio
```

see

Set/GetGearMaster (p. 108), **MultiUpdate** (p. 55), **Update** (p. 167)

SetHoldingCurrent 5Eh
GetHoldingCurrent 5Fh

Syntax

SetHoldingCurrent *axis parameter value*
GetHoldingCurrent *axis parameter*

Motor Types

		Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
parameter	*Holding Motor Limit*	0
	Holding Delay	1

	Type	Range/Scaling
value	unsigned 16-bit	see below

Packet Structure

SetHoldingCurrent

0	*axis*	**5E**h
15 12	11 8	7 0

First Data Word

write | *parameter* | 15 0

Second Data Word

write | *value* | 15 0

GetHoldingCurrent

0	*axis*	**5F**h
15 12	11 8	7 0

First Data Word

write | *parameter* | 15 0

Second Data Word

read | *value* | 15 0

Description

SetHoldingCurrent configures the operation of the holding current. The two parameters to set are *Holding Motor Limit*, the maximum commanded current when in holding, and *Holding Delay*, the number of cycles to wait after end of move before going into holding current.

The *Holding Motor Limit* is in units of % maximum current, with scaling of 100/2^15. Its range is 0 to 2^15–1. It defines the value to which the current will be limited when in the holding state. This limit is applied as an additional limit to the motor limit, so the lower of the two will affect the true limit.

The *Holding Delay* is in units of trajectory generator cycles, with unity scaling and a range of 0 to 2^15–2. It defines the wait time between ending a move and switching to the holding current limit. That is, there will be a delay of *Holding Delay* trajectory cycles after Motion Complete, after which the In Holding bit in the Drive Status register will be set, and the motor command will be limited by the *Holding Motor Limit*. When the *Holding Delay* is set to 2^15–1 (its default), the axis will never go into holding current.

GetHoldingCurrent gets the indicated holding current parameter.

Restrictions

For pulse & direction motor types, only the *Holding Delay* is used. It delays the assertion of the At Rest output by the indicated number of cycles after a move is complete.

C-Motion API

```
PMDresult PMDSetHoldingCurrent(PMDAxisInterface axis_intf,
                               PMDuint16 parameter,
                               PMDuint16 value)
PMDresult PMDGetHoldingCurrent(PMDAxisInterface axis_intf,
                               PMDuint16 parameter,
                               PMDuint16* value)
```

VB-Motion API

```
Dim value as Short
MagellanAxis.HoldingCurrent( parameter ) = value
value = MagellanAxis.HoldingCurrent( parameter )
```

see

GetDriveStatus (p. 38), **Set/GetSampleTime** (p. 140)

SetInterruptMask 2Fh
GetInterruptMask 56h

Syntax

SetInterruptMask *axis mask*
GetInterruptMask *axis*

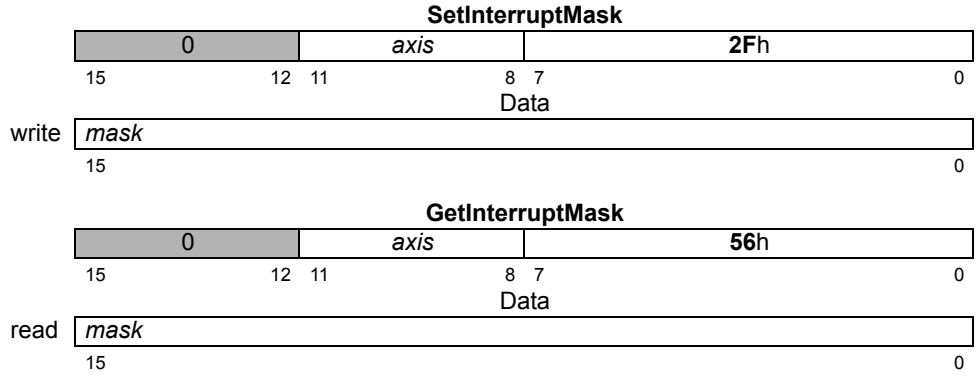
Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
mask	*Motion Complete*	0001h
	Wrap-around	0002h
	Breakpoint 1	0004h
	Capture Received	0008h
	Motion Error	0010h
	Positive Limit	0020h
	Negative Limit	0040h
	Instruction Error	0080h
	Disable	0100h
	Overtemperature Fault	0200h
	Bus Voltage Fault	0400h
	Commutation Error	0800h
	Current Foldback	1000h
	Breakpoint 2	4000h

Packet Structure



Description

SetInterruptMask determines which bits in the Event Status register of the specified ***axis*** will cause a host interrupt. For each interrupt ***mask*** bit that is set to 1, the corresponding Event Status register bit will cause an interrupt when that status register bit goes active (is set to 1). Interrupt mask bits set to 0 will not generate interrupts.

GetInterruptMask returns the *mask* for the specified *axis*.

SetInterruptMask also controls CAN event notification when using the motion processor's CAN 2.0B interface. Whenever a host interrupt is activated, a CAN message is generated using message ID 180h + nodeID, notifying interested CAN nodes of the change in the Event Status register.

Example: The interrupt *mask* value 28h will generate an interrupt when either the Positive Limit bit or the Capture Received bit of the Event Status register goes active (set to 1).

Restrictions

C-Motion API

```
PMDresult PMDSetInterruptMask(PMDAxisInterface axis_intf,
                              PMDuint16 mask)
PMDresult PMDGetInterruptMask(PMDAxisInterface axis_intf,
                              PMDuint16* mask)
```

VB-Motion API

```
Dim mask as Short
MagellanAxis.InterruptMask = mask
mask = MagellanAxis.InterruptMask
```

see **ClearInterrupt** (p. 22), **GetInterruptAxis** (p. 43), **Set/GetFaultOutMask** (p. 104)

SetJerk / GetJerk — buffered — 13h / 58h

Syntax

SetJerk *axis jerk*
GetJerk *axis*

Motor Types

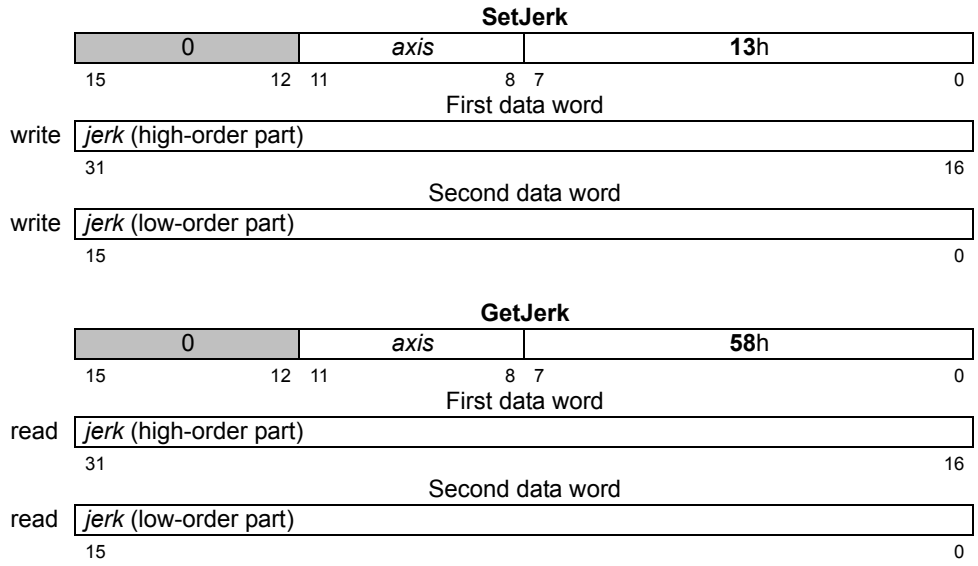
DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

	Type	Range	Scaling	Units
jerk	unsigned 32 bits	0 *to* $2^{31}-1$	$1/2^{32}$	counts/cycle^3 microsteps/cycle^3

Packet Structure



Description

SetJerk loads the Jerk register in the parameter buffer for the specified ***axis***.

GetJerk reads the contents of the Jerk register.

Scaling example: To load a jerk value (rate of change of acceleration) of 0.012345 counts/cycle^3 (or steps/cycle^3) multiply by 2^{32} or 4,294,967,296. In this example this gives a value to load of 53,021,371 (decimal) which corresponds to a high word of 0329h and a low word of 0ABBh when loading each word in hexadecimal.

Restrictions

SetJerk is a buffered command. The value set using this command will not take effect until the next **Update** or **MultiUpdate** command, with the Trajectory Update bit set in the update mask.

This command is used only with the S-curve profile mode. It is not used with the Trapezoidal, Velocity Contouring, or Electronic Gear profile modes.

C-Motion API

```
PMDresult PMDSetJerk(PMDAxisInterface axis_intf, PMDuint32 jerk)
PMDresult PMDGetJerk(PMDAxisInterface axis_intf, PMDuint32* jerk)
```

VB-Motion API

```
Dim jerk as Long
MagellanAxis.Jerk = jerk
jerk = MagellanAxis.Jerk
```

see

Set/GetAcceleration (p. 66), **Set/GetDeceleration** (p. 95), **Set/GetPosition** (p. 133), **Set/GetVelocity** (p. 165), **MultiUpdate** (p. 55), **Update** (p. 167)

SetMotionCompleteMode EBh
GetMotionCompleteMode ECh

Syntax

SetMotionCompleteMode *axis mode*
GetMotionCompleteMode *axis*

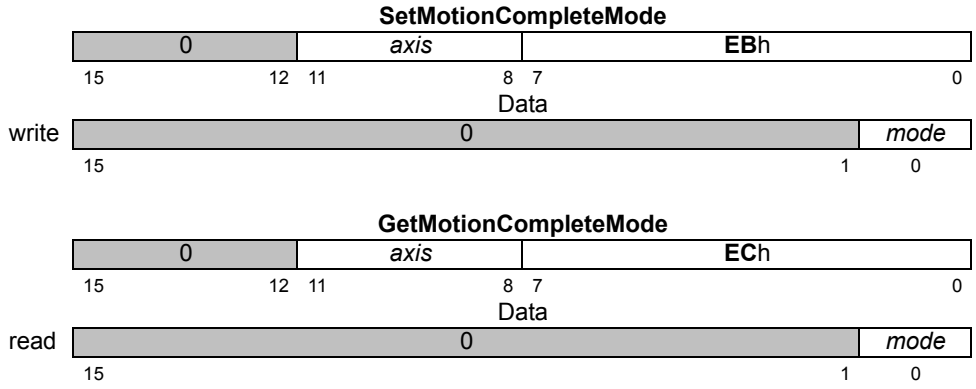
Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
mode	*commanded*	0
	actual	1

Packet Structure



Description

SetMotionCompleteMode establishes the source for the comparison which determines the motion-complete status for the specified ***axis***. When set to ***commanded***, the motion is considered complete when the profile velocity reaches zero (0) and no further motion will occur without an additional host command. This mode is unaffected by the actual encoder location.

When set to ***actual*** mode the motion complete bit will be set when the above condition is true, and when the actual encoder position has been within the settle window (**SetSettleWindow** command) for the number of cycles specified by the **SetSettleTime** command. The settle timer is started at zero (0) at the end of the trajectory profile motion, so a minimum delay of settle time cycles will occur after the trajectory profile motion is complete.

GetMotionCompleteMode returns the value for the motion-complete mode.

Restrictions

C-Motion API

```
PMDresult PMDSetMotionCompleteMode(PMDAxisInterface axis_intf,
                                   PMDuint16 mode)
PMDresult PMDGetMotionCompleteMode(PMDAxisInterface axis_intf,
                                   PMDuint16* mode)
```

VB-Motion API

```
Dim mode as Short
MagellanAxis.MotionCompleteMode = mode
mode = MagellanAxis.MotionCompleteMode
```

see

Set/GetSettleTime (p. 144), **Set/GetSettleWindow** (p. 145)

SetMotorBias 0Fh
GetMotorBias 2Dh

Syntax

SetMotorBias *axis bias*
GetMotorBias *axis*

Motor Types

DC Brush	Brushless DC		

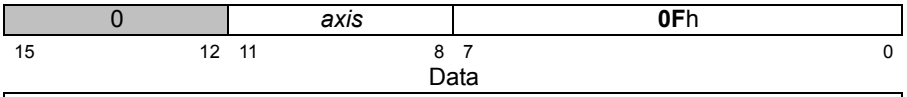
Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

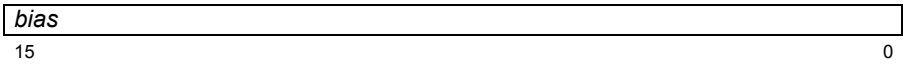
	Type	Range	Scaling	Units
bias	signed 16 bits	-2^{15} *to* $2^{15}-1$	$100/2^{15}$	% output

Packet Structure

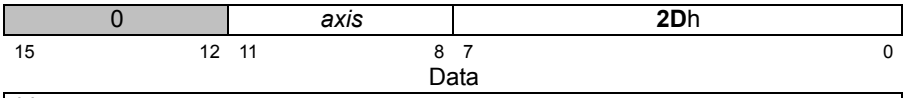
SetMotorBias



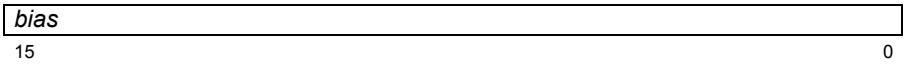
write



GetMotorBias



read



Description

SetMotorBias sets the output ***bias*** of the digital servo filter for the specified ***axis***.

GetMotorBias reads the value of the ***bias*** of the digital servo filter.

Scaling example: If it is desired that a motor bias value of –2.5% of full scale be placed on the servo filter output, then this register should be loaded with a value of –2.5*32,768/100 = –819 (decimal). This corresponds to a loaded hexadecimal value of 0FCCDh.

Restrictions

C-Motion API

```
PMDresult PMDSetMotorBias(PMDAxisInterface axis_intf, PMDint16 bias)
PMDresult PMDGetMotorBias(PMDAxisInterface axis_intf, PMDint16* bias)
```

VB-Motion API

```
Dim bias as Short
MagellanAxis.MotorBias = bias
bias = MagellanAxis.MotorBias
```

see

Set/GetMotorCommand (p. 118), **Set/GetMotorLimit** (p. 119)

Syntax

SetMotorCommand *axis command*
GetMotorCommand *axis*

Motor Types

DC Brush	Brushless DC	Microstepping	

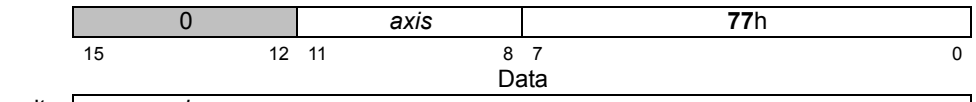
Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

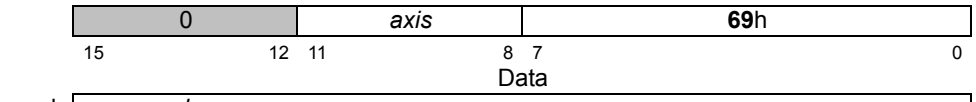
	Type	Range	Scaling	Units
command	signed 16 bits	-2^{15} *to* $2^{15}-1$	$100/2^{15}$	% output

Packet Structure

SetMotorCommand



GetMotorCommand



Description

SetMotorCommand loads the Motor Command buffer register of the specified ***axis***. For axes configured for microstepping motors, this command is used to control the magnitude of the output waveform. For DC brush and brushless DC motors, this command directly sets the Motor Output register when the Position Loop and Trajectory Generator modules are disabled in the operating mode.

GetMotorCommand reads the contents of the motor command buffer register.

Scaling example: If it is desired that a Motor Command value of 13.7% of full scale be output to the motor, then this register should be loaded with a value of 13.7 * 32,768/100 = 4,489 (decimal). This corresponds to a hexadecimal value of 1189h.

Restrictions

SetMotorCommand is a buffered command. The value set using this command will not take effect until the next **Update** or **MultiUpdate** command, with the Position Loop Update bit set in the update mask.

C-Motion API

```
PMDresult PMDSetMotorCommand(PMDAxisInterface axis_intf,
                             PMDint16 command)
PMDresult PMDGetMotorCommand(PMDAxisInterface axis_intf,
                             PMDint16* command)
```

VB-Motion API

```
Dim command as Short
MagellanAxis.MotorCommand = command
command = MagellanAxis.MotorCommand
```

see

Set/GetMotorBias (p. 117), **Set/GetMotorLimit** (p. 119), **Set/GetOperatingMode** (p. 122), **MultiUpdate** (p. 55), **Update** (p. 167)

SetMotorLimit 06h
GetMotorLimit 07h

Syntax

SetMotorLimit *axis limit*
GetMotorLimit *axis*

Motor Types

DC Brush	Brushless DC		

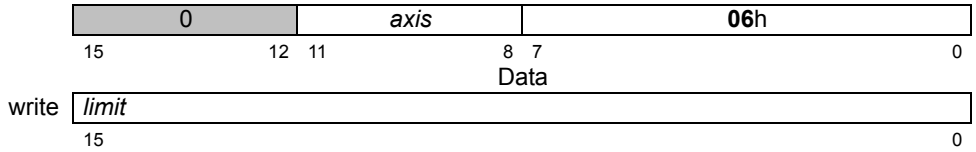
Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

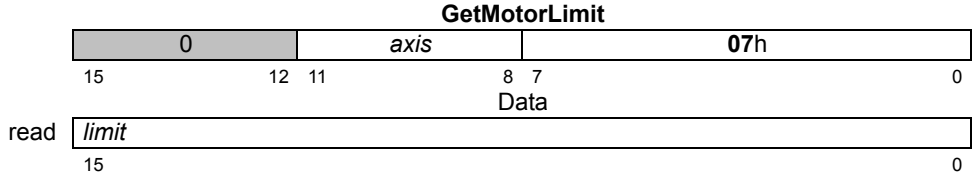
	Type	Range	Scaling	Units
limit	unsigned 16 bits	0 *to* $2^{15}-1$	$100/2^{15}$	% output

Packet Structure

SetMotorLimit



GetMotorLimit



Description

SetMotorLimit sets the maximum value for the motor output command allowed by the digital servo filter of the specified ***axis***. Motor command values beyond this value will be clipped to the specified motor command limit. For example if the motor limit was set to 1,000 and the servo filter determined that the current motor output value should be 1,100, the actual output value would be 1,000. Conversely, if the output value was –1,100, then it would be clipped to –1,000. This command is useful for protecting amplifiers, motors, or system mechanisms when it is known that a motor command exceeding a certain value will cause damage.

GetMotorLimit reads the motor limit value.

Scaling example: If it is desired that a motor limit of 75% of full scale be established, then this register should be loaded with a value of 75.0 * 32,767/100 = 24,576 (decimal). This corresponds to a hexadecimal value of 06000h.

Restrictions

This command only affects the motor output when the position loop or trajectory generator is enabled. When the motion processor is in open loop mode, this command has no effect.

C-Motion API

```
PMDresult PMDSetMotorLimit(PMDAxisInterface axis_intf,
                           PMDuint16 limit);
PMDresult PMDGetMotorLimit(PMDAxisInterface axis_intf,
                           PMDuint16* limit)
```

VB-Motion API

```
Dim limit as Short
MagellanAxis.MotorLimit = limit
limit = MagellanAxis.MotorLimit
```

see

Set/GetMotorBias (p. 117), **Set/GetMotorCommand** (p. 118), **Set/GetOperatingMode** (p. 122)

SetMotorType 02h
GetMotorType 03h

Syntax

SetMotorType *axis type*
GetMotorType *axis*

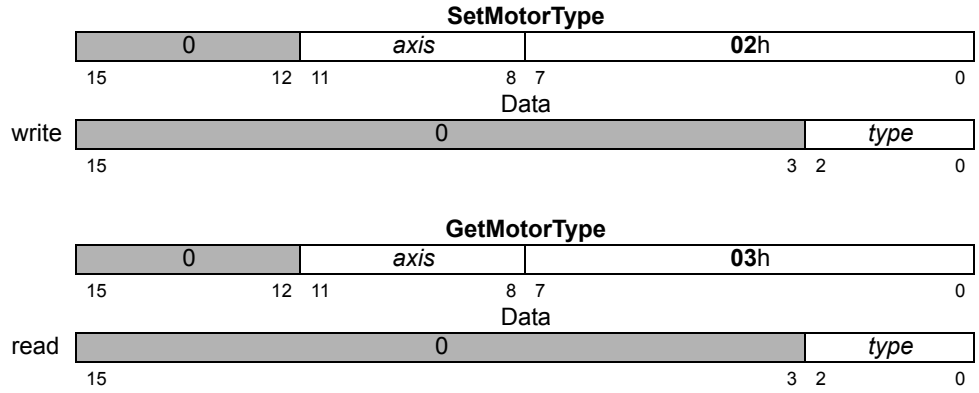
Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
type	*Brushless DC (3 phase)*	0
	Brushless DC (2 phase)	1
	Microstepping (3 phase)	2
	Microstepping (2 phase)	3
	Pulse & Direction	4
	DC Brush	7

Packet Structure



Description

SetMotorType sets type of motor being driven by the selected ***axis***. This operation sets the number of phases for commutation on the axis, as well as internally configuring the motion processor for the motor type.

The following table describes each motor type, and the number of phases to be commutated.

Motor type	Commutation
Brushless DC (3 phase)	3 phase
Brushless DC (2 phase)	2 phase
Microstepping (3 phase)	3 phase
Microstepping (2 phase)	2 phase
Pulse & Direction	None
DC Brush	None

GetMotorType returns the configured motor type for the selected ***axis***.

Restrictions

The motor type should only be set once for each axis, either via the motor configuration word during device startup, or immediately after reset using **SetMotorType**. Once it has been set, it should not be changed. Executing **SetMotorType** will reset many variables to their motor type specific default values.

Not all motor types are available on all products. See the product user's guide.

C-Motion API

```
PMDresult PMDSetMotorType(PMDAxisInterface axis_intf, PMDuint8 type)
PMDresult PMDGetMotorType(PMDAxisInterface axis_intf, PMDuint8* type)
```

VB-Motion API

```
Dim type as Short
MagellanAxis.MotorType = type
type = MagellanAxis.MotorType
```

see **Reset** (p. 60)

SetOperatingMode 65h
GetOperatingMode 66h

Syntax

SetOperatingMode *axis mode*
GetOperatingMode *axis*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

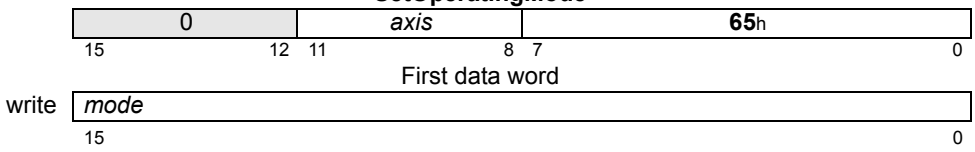
Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

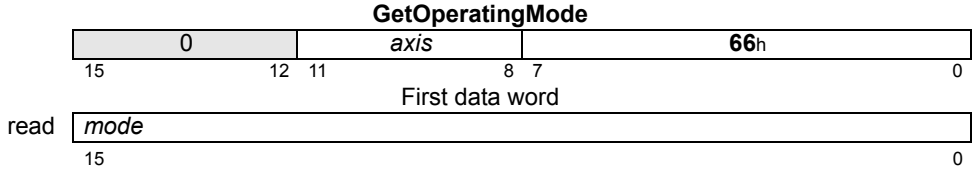
	Type	Range/Scaling
mode	unsigned 16-bit	see below

Packet Structure

SetOperatingMode



GetOperatingMode



Description

SetOperatingMode configures the operating mode of the ***axis***. Each bit of the ***mode*** configures whether a feature/loop of the ***axis*** is active or disabled, as follows:

Name	Bit	Description
Axis Enabled	0	0: No *axis* processing, *axis* outputs in reset state. 1: *axis* active.
Motor Output Enabled	1	0: *axis* motor outputs disabled. 1: *axis* motor outputs enabled.
Current Control Enabled	2	0: *axis* current control bypassed. 1: *axis* current control active.
—	3	Reserved
Position Loop Enabled	4	0: *axis* position loop bypassed. 1: *axis* position loop active.
Trajectory Enabled	5	0: trajectory generator disabled. 1: trajectory generator enabled.
—	6–15	Reserved

When the ***axis*** is disabled, no processing will be done on the axis, and the axis outputs will be at their reset states. When the axis motor output is disabled, the axis will function normally, but its motor outputs will be in their disabled state. When a loop is disabled (position or current loop), it operates by passing its input directly to its output, and clearing all internal state variables (such as integrator sums, etc.). When the trajectory generator is disabled, it operates by commanding 0 velocity.

Description (cont.)

For example, to configure an axis for Torque mode, (trajectory and position loop disabled) the operating mode would be set to hexadecimal 0007h.

This command should be used to configure the static operating mode of the ***axis***. The actual current operating mode may be changed by the axis in response to safety events, or user-programmable events. In this case, the present operating mode is available using **GetActiveOperatingMode**. **GetOperatingMode** will always return the static operating mode set using **SetOperatingMode**. Executing the **SetOperatingMode** command sets both the static operating mode and the active operating mode to the desired state.

GetOperatingMode gets the operating mode of the ***axis***.

Restrictions

The possible operating modes of an axis is product specific, and in some cases axis specific. See the product user's guide for a description of what operating modes are supported on each axis.

C-Motion API

```
PMDresult PMDSetOperatingMode(PMDAxisInterface axis_intf,
                              PMDuint16 mode)
PMDresult PMDGetOperatingMode(PMDAxisInterface axis_intf,
                              PMDuint16* mode)
```

VB-Motion API

```
Dim mode as Short
MagellanAxis.OperatingMode = mode
mode = MagellanAxis.OperatingMode
```

see

GetActiveOperatingMode (p. 25), **RestoreOperatingMode** (p. 65)

SetOutputMode E0h
GetOutputMode 6Eh

Syntax

SetOutputMode *axis mode*
GetOutputMode *axis*

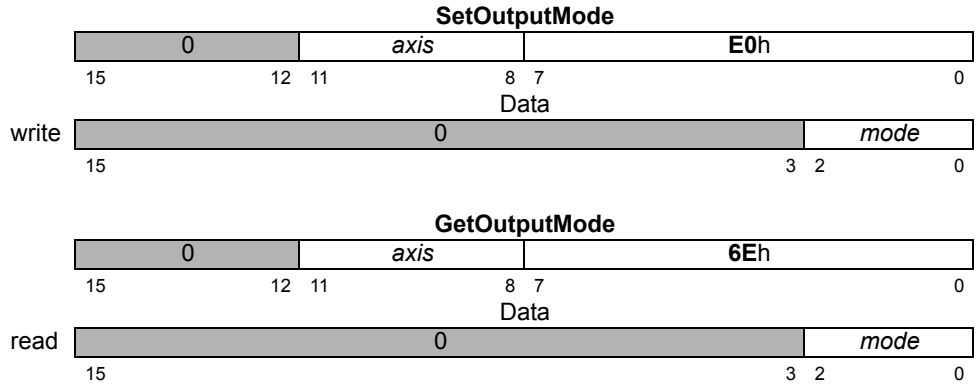
Motor Types

DC Brush	Brushless DC	Microstepping	

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
mode	*Parallel DAC Offset Binary*	0
	PWM Sign Magnitude	1
	PWM 50/50 Magnitude	2
	SPI DAC Offset Binary	3
	Parallel DAC Sign Magnitude	4
	SPI DAC 2's Complement	5
	PWM High/Low	7

Packet Structure



Description

SetOutputMode sets the form of the motor output signal of the specified *axis*.

GetOutputMode returns the value for the motor output mode.

Restrictions

Not all output modes are available on all products. See the product user's guide.

C-Motion API

```
PMDresult PMDSetOutputMode(PMDAxisInterface axis_intf, PMDuint16 mode)
PMDresult PMDGetOutputMode(PMDAxisInterface axis_intf, PMDuint16* mode)
```

VB-Motion API

```
Dim mode as Short
MagellanAxis.OutputMode = mode
mode = MagellanAxis.OutputMode
```

see

Syntax

SetOvertemperatureLimit *axis limit*
GetOvertemperatureLimit *axis*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

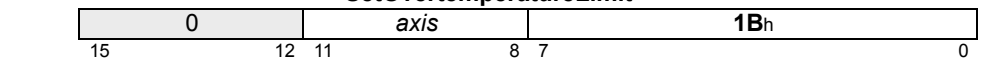
Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

	Type	Range	Scaling	Units
limit	signed 16 bits	–2^15 *to* 2^15–1	2^8	°C

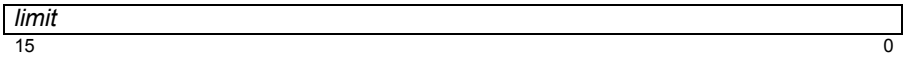
Packet Structure

SetOvertemperatureLimit

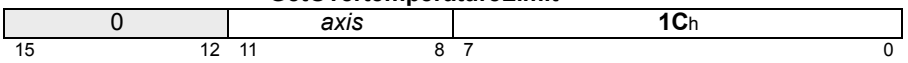


First data word

write

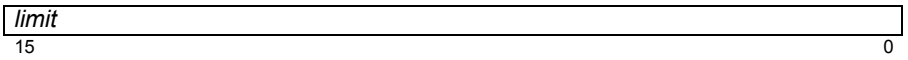


GetOvertemperatureLimit



First data word

read



Description

SetOvertemperatureLimit sets the temperature threshold upon which an overtemperature condition will occurr. For example, to set the overtemperature threshold at 60 degrees C, the value should be 60*256 = 15360. When the programmed threshold is exceeded, the Overtemperature Fault bit is set in the Event Status register, and the ***axis*** enters the overtemperature state.

GetOvertemperatureLimit gets the current overtemperature threshold setting.

Restrictions

Get/SetOvertemperatureLimit is only available in products equipped with temperature sensors.

If the ***axis*** has more than one temperature sensor, the temperature used to compare to the overtemperature threshold will be the highest value of all sensor readings.

The overtemperature threshold cannot be set to a value greater than the reset default setting.

C-Motion API

```
PMDresult PMDSetOvertemperatureLimit (PMDAxisInterface axis_intf,
                                      PMDint16 limit)
PMDresult PMDGetOvertemperatureLimit (PMDAxisInterface axis_intf,
                                      PMDint16* limit)
```

VB-Motion API

```
Dim limit as Short
MagellanAxis.OvertemperatureLimit = limit
limit = MagellanAxis.OvertemperatureLimit
```

see

GetTemperature (p. 48), **GetEventStatus** (p. 39), **ResetEventStatus** (p. 64)

SetPhaseAngle 84h
GetPhaseAngle 2Ch

Syntax

SetPhaseAngle *axis angle*
GetPhaseAngle *axis*

Motor Types

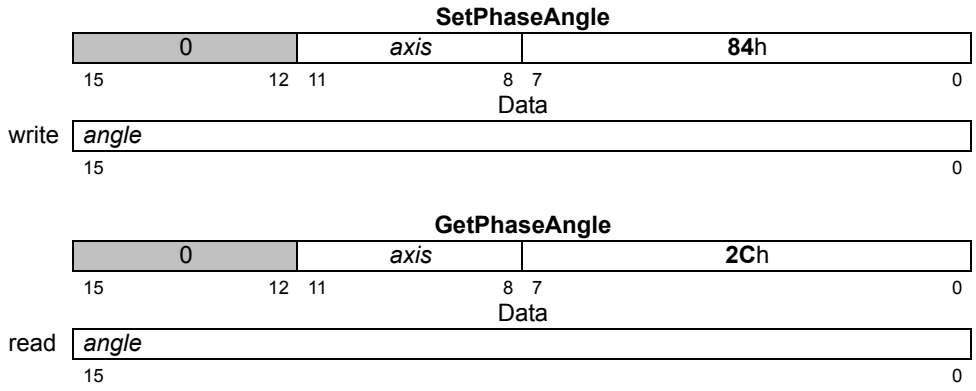
	Brushless DC	Microstepping	

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

	Type	Range	Scaling	Units
angle	unsigned 16 bits	0 *to* $2^{15}-1$	unity	counts microsteps

Packet Structure



Description

SetPhaseAngle sets the instantaneous commutation angle for the specified ***axis***. For brushless DC motors, the phase angle is specified in units of encoder counts. For microstepping motors, it is specified in units of microsteps. **GetPhaseAngle** returns the value of the phase angle. To convert counts to an actual phase angle, divide by the number of encoder counts per electrical cycle and multiply by 360.

For example, if a value of 500 is retrieved using **GetPhaseAngle** and the counts per electrical cycle value has been set to 2,000 (**SetPhaseCounts** command), this corresponds to an angle of (500/2,000)*360 = 90 degrees current phase angle position. **SetPhaseAngle** resets the phase offset previously set by **SetPhaseOffset**.

Restrictions

The specified angle must not exceed the number set by the **SetPhaseCounts** command.

C-Motion API

```
PMDresult PMDSetPhaseAngle(PMDAxisInterface axis_intf,
                           PMDuint16 angle)
PMDresult PMDGetPhaseAngle(PMDAxisInterface axis_intf,
                           PMDuint16* angle)
```

VB-Motion API

```
Dim angle as Short
MagellanAxis.PhaseAngle = angle
angle = MagellanAxis.PhaseAngle
```

see

Set/GetPhaseCounts (p. 128)

SetPhaseCorrectionMode E8h
GetPhaseCorrectionMode E9h

Syntax

SetPhaseCorrectionMode *axis mode*
GetPhaseCorrectionMode *axis*

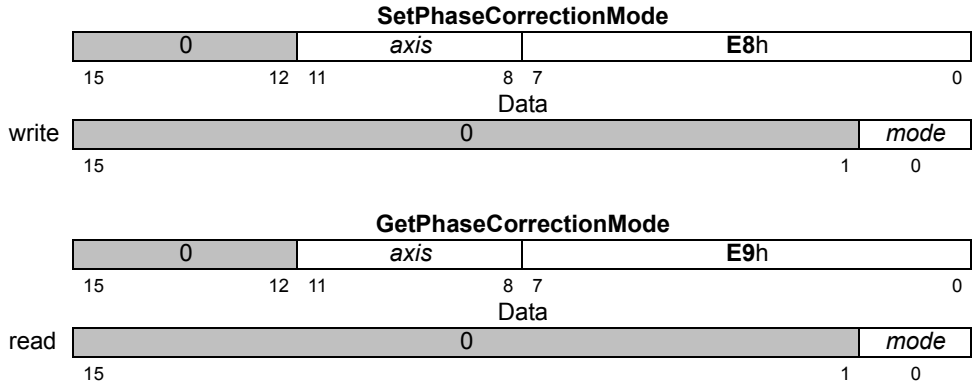
Motor Types

	Brushless DC		

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
mode	*Disable*	0
	Enable	1

Packet Structure



Description

SetPhaseCorrectionMode sets the phase correction mode for the specified ***axis*** to either 0 (disabled) or 1 (enabled). When phase correction is enabled, the encoder Index signal is used to update the commutation phase angle once per motor revolution. This ensures that the commutation angle will remain correct even if some encoder counts are lost due to electrical noise, or due to the number of encoder counts per electrical phase not being an integer.

GetPhaseCorrectionMode returns the phase correction mode.

Restrictions

C-Motion API

```
PMDresult PMDSetPhaseCorrectionMode(PMDAxisInterface axis_intf,
                                    PMDuint16 mode)
PMDresult PMDGetPhaseCorrectionMode(PMDAxisInterface axis_intf,
                                    PMDuint16* mode)
```

VB-Motion API

```
Dim mode as Short
MagellanAxis.PhaseCorrectionMode = mode
mode = MagellanAxis.PhaseCorrectionMode
```

see

GetPhaseCommand (p. 44), **InitializePhase** (p. 54), **Set/GetPhaseCounts** (p. 128)

SetPhaseCounts 75h
GetPhaseCounts 7Dh

Syntax

SetPhaseCounts *axis counts*
GetPhaseCounts *axis*

Motor Types

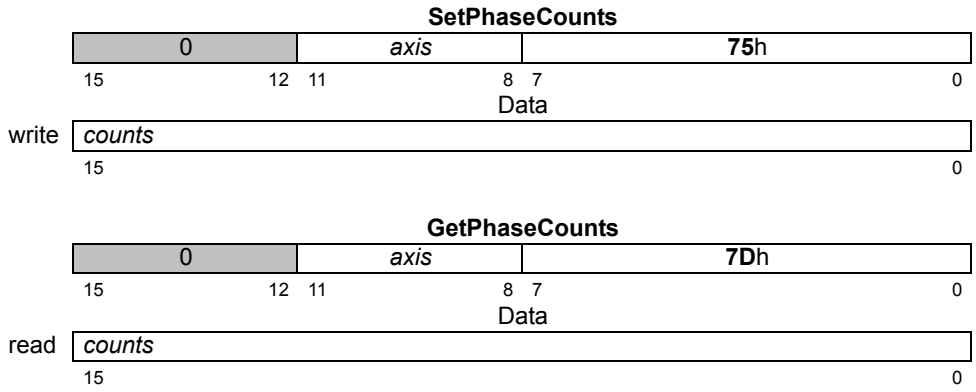
	Brushless DC	Microstepping	

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

	Type	Range	Scaling	Units
counts	unsigned 16 bits	1 *to* $2^{15}-1$	unity	counts microsteps

Packet Structure



Description

For axes configured for brushless DC motor types, **SetPhaseCounts** sets the number of encoder counts per electrical cycle of the motor. The number of electrical cycles is equal to 1⁄2 the number of motor poles. If this value is not an integer, then the closest integer value should be used, and phase correction mode should be enabled. See **SetPhaseCorrectionMode** (p. 127).

For axes configured for microstepping motor types, the number of microsteps per full step is set using the **SetPhaseCounts** command. The parameter used for this command represents the number of microsteps per electrical cycle (4 times the desired number of microsteps). For example, to set 64 microsteps per full step, the **SetPhaseCounts 256** command should be used. The maximum number of microsteps that can be generated per full step is 256, giving a maximum parameter value of 1024.

GetPhaseCounts returns the number of counts or microsteps per electrical cycle.

Restrictions

C-Motion API

```
PMDresult PMDSetPhaseCounts(PMDAxisInterface axis_intf, PMDuint16
counts)
PMDresult PMDGetPhaseCounts(PMDAxisInterface axis_intf, PMDuint16*
counts)
```

VB-Motion API

```
Dim counts as Short
MagellanAxis.PhaseCounts = counts
counts = MagellanAxis.PhaseCounts
```

see

Set/GetPhaseAngle (p. 126)

SetPhaseInitializeMode E4h
GetPhaseInitializeMode E5h

Syntax

SetPhaseInitializeMode *axis mode*
GetPhaseInitializeMode *axis*

Motor Types

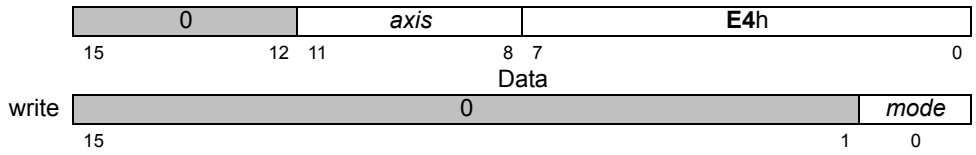
	Brushless DC		

Arguments

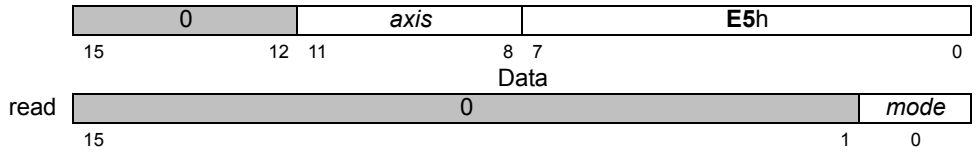
Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
mode	*Algorithmic*	0
	Hall-based	1

Packet Structure

SetPhaseInitializeMode



GetPhaseInitializeMode



Description

SetPhaseInitializeMode establishes the mode in which the specified ***axis*** is to be initialized for commutation. The options are ***Algorithmic*** and ***Hall-based***. In algorithmic mode the motion processor briefly stimulates the motor windings and sets the initial phasing based on the observed motor response. In Hall-based initialization mode, the three Hall sensor signals are used to determine the motor phasing.

GetPhaseInitializeMode returns the value of the initialization mode.

Restrictions

Algorithmic mode should only be selected if it is known that the axis is free to move in both directions, and that a brief uncontrolled move can be tolerated by the motor, mechanism, and load.

C-Motion API

```
PMDresult PMDSetPhaseInitializeMode(PMDAxisInterface axis_intf,
                                    PMDuint16 mode)
PMDresult PMDGetPhaseInitializeMode(PMDAxisInterface axis_intf,
                                    PMDuint16* mode)
```

VB-Motion API

```
Dim mode as Short
MagellanAxis.PhaseInitializeMode = mode
mode = MagellanAxis.PhaseInitializeMode
```

see

InitializePhase (p. 54), **Set/GetPhaseInitializeTime** (p. 130)

SetPhaseInitializeTime 72h
GetPhaseInitializeTime 7Ch

Syntax

SetPhaseInitializeTime *axis time*
GetPhaseInitializeTime *axis*

Motor Types

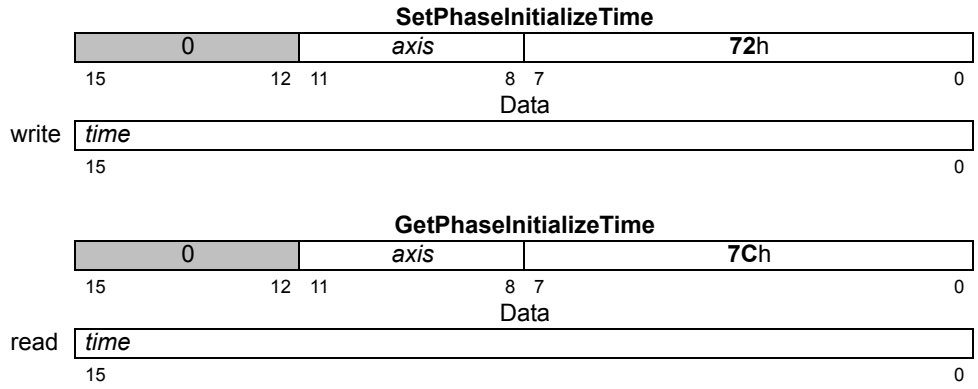
	Brushless DC		

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

	Type	Range	Scaling	Units
time	unsigned 16 bits	0 *to* $2^{15}-1$	unity	cycles

Packet Structure



Description

SetPhaseInitializeTime sets the time value (in cycles) to be used during the algorithmic phase initialization procedure. This value determines the duration of each of the four segments in the phase initialization algorithm.

GetPhaseInitializeTime returns the value of the phase initialization time.

Restrictions

C-Motion API

```
PMDresult PMDSetPhaseInitializeTime(PMDAxisInterface axis_intf,
                                    PMDuint16 time)
PMDresult PMDGetPhaseInitializeTime(PMDAxisInterface axis_intf,
                                    PMDuint16* time)
```

VB-Motion API

```
Dim time as Short
MagellanAxis.PhaseInitializeTime = time
time = MagellanAxis.PhaseInitializeTime
```

see

InitializePhase (p. 54), **Set/GetPhaseInitializeMode** (p. 129)

Syntax

SetPhaseOffset *axis offset*
GetPhaseOffset *axis*

Motor Types

	Brushless DC		

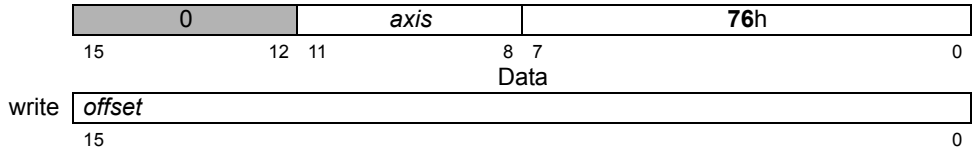
Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

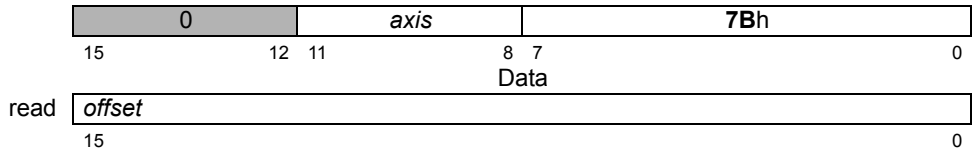
	Type	Range	Scaling	Units
offset	unsigned 16 bits	0 *to* 2^{15}–1	unity	counts

Packet Structure

SetPhaseOffset



GetPhaseOffset



Description

SetPhaseOffset sets the offset from the index mark of the specified ***axis*** to the internal zero phase angle. This command will have no immediate effect on the commutation angle but will have an effect once the index pulse is encountered. The settable range of phase offset is 0 to 32,767.

GetPhaseOffset returns the value of the phase offset.

To convert counts to a phase angle in degrees, divide by the number of encoder counts per electrical cycle and multiply by 360. For example, if a value of 500 is specified using **SetPhaseOffset** and the counts per electrical cycle value has been set to 2,000 (**SetPhaseCounts** command) this corresponds to an angle of (500/2,000)*360 = 90 degrees phase angle at the index mark.

Restrictions

Before the first index capture has occurred, **GetPhaseOffset** will return –1.

C-Motion API

```
PMDresult PMDSetPhaseOffset(PMDAxisInterface axis_intf,
                            PMDint16 offset)
PMDresult PMDGetPhaseOffset(PMDAxisInterface axis_intf,
                            PMDint16* offset)
```

VB-Motion API

```
Dim offset as Short
MagellanAxis.PhaseOffset = offset
offset = MagellanAxis.PhaseOffset
```

see

SetPhasePrescale E6h
GetPhasePrescale E7h

Syntax

SetPhasePrescale *axis scale*
GetPhasePrescale *axis*

Motor Types

	Brushless DC		

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
scale	*Off*	0
	1/64	1
	1/128	2
	1/256	3

Packet Structure

Description

SetPhasePrescale controls scaling of the encoder counts before they are used to calculate a commutation angle for the specified ***axis***. When operated in the pre-scale mode, the motion processor can commutate motors with a high number of counts per electrical cycle, such as motors with very high accuracy encoders.

SetPhasePrescale Off removes the scale factor.

GetPhasePrescale returns the value of the scaling mode.

Restrictions

C-Motion API

```
PMDresult PMDSetPhasePrescale(PMDAxisInterface axis_intf,
                              PMDuint16 scale);
PMDresult PMDGetPhasePrescale(PMDAxisInterface axis_intf,
                              PMDuint16* scale)
```

VB-Motion API

```
Dim scale as Short
MagellanAxis.PhasePrescale = scale
scale = MagellanAxis.PhasePrescale
```

see

Syntax

SetPosition *axis position*
GetPosition *axis*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

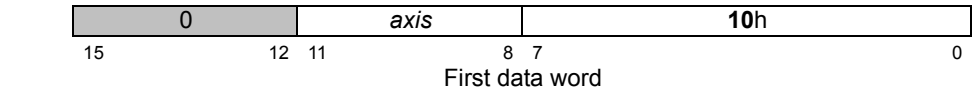
Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

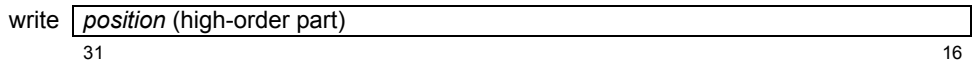
	Type	Range	Scaling	Units
position	signed 32 bits	-2^{31} *to* $2^{31}-1$	unity	counts microsteps

Packet Structure

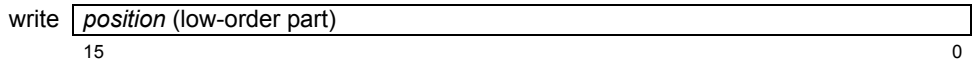
SetPosition



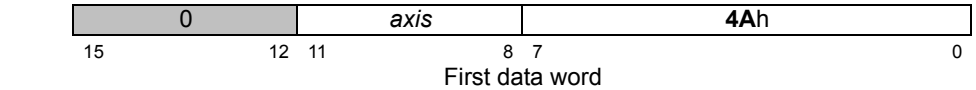
First data word



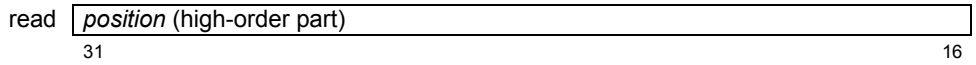
Second data word



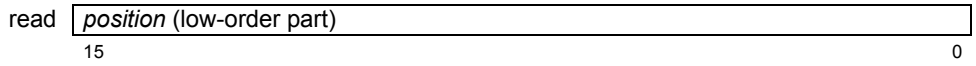
GetPosition



First data word



Second data word



Description

SetPosition specifies the trajectory destination of the specified ***axis***. It is used in the Trapezoidal and S-curve profile modes.

GetPosition reads the contents of the buffered position register.

Restrictions

SetPosition is a buffered command. The value set using this command will not take effect until the next **Update** or **MultiUpdate** command, with the Trajectory Update bit set in the update mask.

C-Motion API

```
PMDresult PMDSetPosition(PMDAxisInterface axis_intf,
                         PMDint32 position);
PMDresult PMDGetPosition(PMDAxisInterface axis_intf,
                         PMDint32* position)
```

VB-Motion API

```
Dim position as Long
MagellanAxis.Position = position
position = MagellanAxis.Position
```

see

Set/GetAcceleration (p. 66), **Set/GetDeceleration** (p. 95), **Set/GetJerk** (p. 115), **Set/GetVelocity** (p. 165), **MultiUpdate** (p. 55), **Update** (p. 167)

SetPositionErrorLimit 97h
GetPositionErrorLimit 98h

Syntax

SetPositionErrorLimit *axis limit*
GetPositionErrorLimit *axis*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

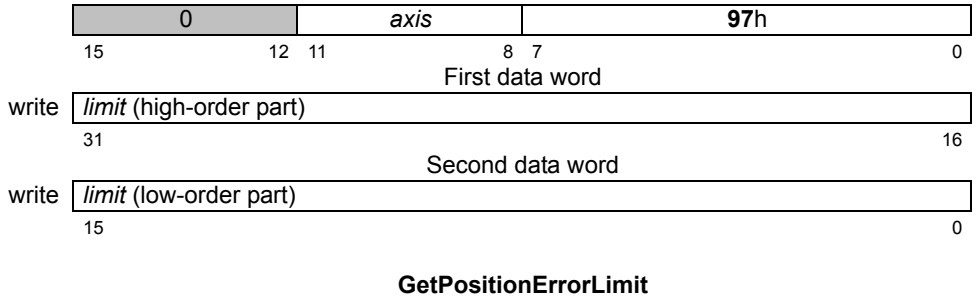
Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

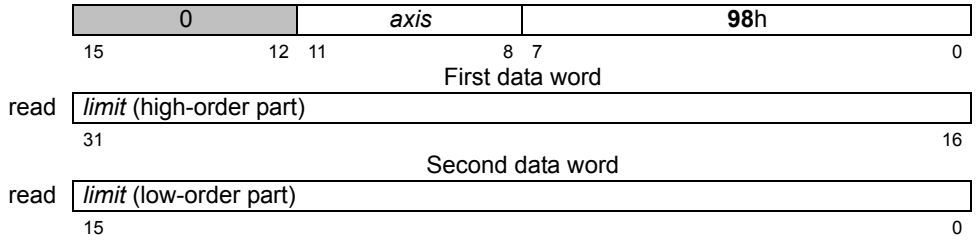
	Type	Range	Scaling	Units
limit	unsigned 32 bits	0 *to* $2^{31}-1$	unity	counts microsteps

Packet Structure

SetPositionErrorLimit



GetPositionErrorLimit



Description

SetPositionErrorLimit sets the absolute value of the maximum position error allowable by the motion processor for the specified ***axis***. If the position error exceeds this ***limit***, a motion error occurs. Such a motion error can cause a choice of actions, or no action, configurable using the **SetEventAction** (Motion Error) command.

When the motor type is microstepping or pulse & direction, this value is set in microsteps or steps, respectively.

GetPositionErrorLimit returns the value of the position error limit.

Restrictions

C-Motion API

```
PMDresult PMDSetPositionErrorLimit(PMDAxisInterface axis_intf,
                                   PMDuint32 limit)
PMDresult PMDGetPositionErrorLimit(PMDAxisInterface axis_intf,
                                   PMDuint32* limit)
```

VB-Motion API

```
Dim limit as Long
MagellanAxis.PositionErrorLimit = limit
limit = MagellanAxis.PositionErrorLimit
```

see

GetPositionError (p. 45), **GetActualPosition** (p. 70), **Set/GetPosition** (p. 133), **Set/GetEventAction** (p. 102)

Syntax

SetPositionLoop *axis parameter value*
GetPositionLoop *axis parameter*

Motor Types

DC Brush	Brushless DC		

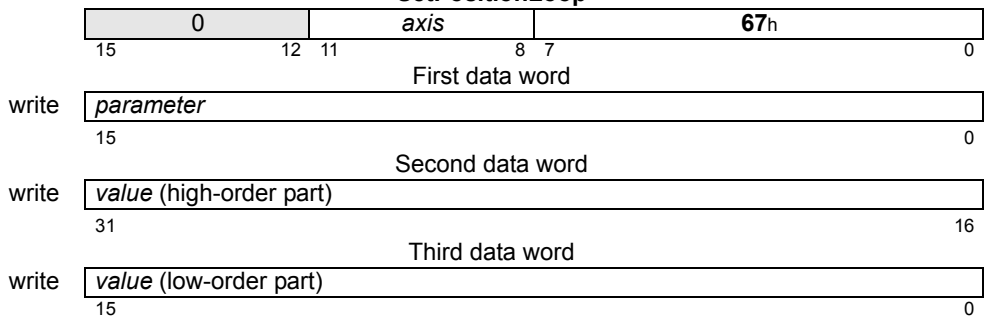
Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
parameter	*PID Proportional Gain (Kp)*	0
	PID Integrator Gain (Ki)	1
	PID Integrator Limit (Ilimit)	2
	PID Derivative Gain (Kd)	3
	PID Derivative Time	4
	PID Output Gain (Kout)	5
	Velocity Feedforward Gain (Kvff)	6
	Acceleration Feedforward Gain (Kaff)	7
	Biquad1, Enable Filter	8
	Biquad1, CoefficientB0	9
	Biquad1, CoefficientB1	10
	Biquad1, CoefficientB2	11
	Biquad1, CoefficientA1	12
	Biquad1, CoefficientA2	13
	Biquad1, CoefficientK	14
	Biquad2, Enable filter	15
	Biquad2, CoefficientB0	16
	Biquad2, CoefficientB1	17
	Biquad2, CoefficientB2	18
	Biquad2, CoefficientA1	19
	Biquad2, CoefficientA2	20
	Biquad2, CoefficientK	21

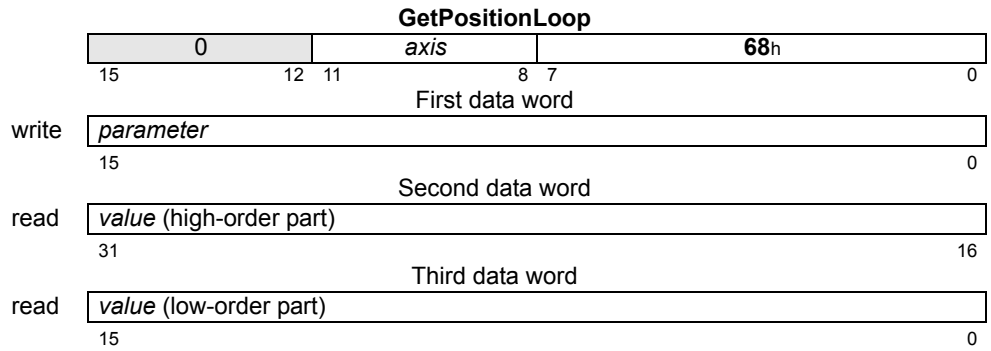
	Type	Range/Scaling
value	signed 32 bits	see below

Packet Structure

SetPositionLoop



Packet Structure (cont.)



Description

Set/GetPositionLoop is used to configure the operating parameters of the PID position loop. See the product user's guide for more information on how each ***parameter*** is used in the position loop processing. Though these commands always use 32-bit data, the range and format vary depending on the ***parameter***, as follows:

Parameter	Range	Scaling	Units
Velocity Feedforward Gain (Kvff)	0 to 2^15–1	unity	gain/cycles
Acceleration Feedforward Gain (Kaff)	0 to 2^15–1	unity	gain/cycles^2
PID Proportional Gain (Kp)	0 to 2^15–1	unity	gain
PID Integrator Gain (Ki)	0 to 2^15–1	1/256	gain/cycles
PID Derivative Gain (Kd)	0 to 2^15–1	unity	gain*cycles
PID Integrator Limit (Ilimit)	0 to 2^31–1	unity	count*cycles
PID Derivative Time	1 to 2^15–1	unity	cycles
PID Output Gain (Kout)	0 to 2^16–1	100/2^16	% output
Biquad1, Enable Filter	0 to 1	0=disable, 1=enable	
Biquad1, CoefficientB0	–2^15 to 2^15–1	unity	
Biquad1, CoefficientB1	–2^15 to 2^15–1	unity	
Biquad1, CoefficientB2	–2^15 to 2^15–1	unity	
Biquad1, CoefficientA1	–2^15 to 2^15–1	unity	
Biquad1, CoefficientA2	–2^15 to 2^15–1	unity	
Biquad1, CoefficientK	0 to 2^15–1	unity	
Biquad2, Enable Filter	0 to 1	0=disable, 1=enable	
Biquad2, CoefficientB0	–2^15 to 2^15–1	unity	
Biquad2, CoefficientB1	–2^15 to 2^15–1	unity	
Biquad2, CoefficientB2	–2^15 to 2^15–1	unity	
Biquad2, CoefficientA1	–2^15 to 2^15–1	unity	
Biquad2, CoefficientA2	–2^15 to 2^15–1	unity	
Biquad2, CoefficientK	0 to 2^15–1	unity	

Many of these parameters are self-descriptive. However, below are some additional comments on the use of specific parameters.

- *PID Derivative Time* has units of cycles. This is the sample time of the ***axis***, as configured by **SetSampleTime**. For example, if set to 10, the derivative term will be computed every 10 cycles of the axis position loop. *PID Integrator Limit* has units of count*cycles, and scaling of unity. This matches the units and scaling of the position loop integrator sum. For example, a constant position error of 100 counts which is present for 256 cycles will result an an integrator sum of 100*256 = 25,600.
- *PID Integrator Gain* has scaling of 1/256. Thus, a setting of 256 corresponds to "unity" integrator gain. From the above example, this would make the integrator sum of 25,600 create a contribution to the PID output of 25,600.
- *PID Output Gain* is a scaling factor applied to the output of the digital servo filter, with units of % output. Its default value is 65,535, or approximately 100% output. To set the scaling to, for example, 50% of output, *PID Output Gain* would be set to 32,767.
- The biquad coefficients configure the two biquad output filters. If both filters are enabled, their outputs are chained (filter1 followed by filter2). If filter1 is disabled for an axis, filter2 is also disabled for that axis, regardless of user setting of *Biquad2 Enable Filter*. The signed coefficients and unsigned scalar K combine to implement the following equation, for each filter:

$$Y_n = K \times (B_0 \times X_n + B_1 \times X_{n-1} + B_2 \times X_{n-2} + A_1 \times Y_{n-1} \times A_2 \times Y_{n-2})$$

Where Y_n is the filter output at cycle n, and X_n is the filter input at cycle n.

Restrictions

Set/GetPositionLoop are buffered commands. All parameters set are buffered, and will not take effect until an update is done on the position loop (through **Update** command, **MultiUpdate** command, or update action on breakpoint). The values read by **GetPositionLoop** are the buffered settings.

C-Motion API

```
PMDresult PMDSetPositionLoop(PMDAxisInterface axis_intf,
                             PMDuint16 parameter,
                             PMDint32 value)
PMDresult PMDGetPositionLoop(PMDAxisInterface axis_intf,
                             PMDuint16 parameter,
                             PMDint32* value)
```

VB-Motion API

```
Dim value as Long
MagellanAxis.PositionLoop( parameter ) = value
value = MagellanAxis.PositionLoop( parameter )
```

see

Update (p. 167), **Set/GetUpdateMask** (p. 163), **MultiUpdate** (p. 55), **Set/GetBreakpointUpdateMask** (p. 77), **GetPositionLoopValue** (p. 46)

SetProfileMode buffered A0h
GetProfileMode A1h

Syntax

SetProfileMode *axis mode*
GetProfileMode *axis*

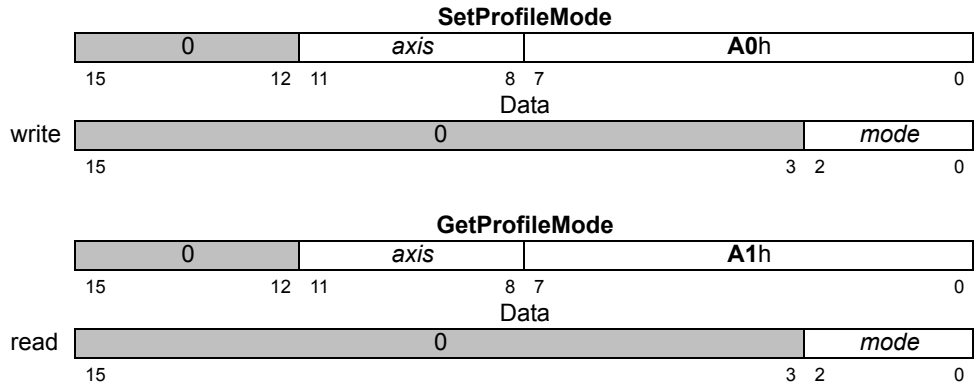
Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
mode	*Trapezoidal*	0
	Velocity Contouring	1
	S-curve	2
	Electronic Gear	3

Packet Structure



Description

SetProfileMode sets the profile mode for the specified ***axis***.

GetProfileMode returns the contents of the profile-mode register for the specified ***axis***.

Restrictions

SetProfileMode is a buffered command. The value set using this command will not take effect until the next **Update** or **MultiUpdate** command, with the Trajectory Update bit set in the update mask.

C-Motion API

```
PMDresult PMDSetProfileMode(PMDAxisInterface axis_intf,
                            PMDuint16 mode)
PMDresult PMDGetProfileMode(PMDAxisInterface axis_intf,
                            PMDuint16* mode)
```

VB-Motion API

```
Dim mode as Short
MagellanAxis.ProfileMode = mode
mode = MagellanAxis.ProfileMode
```

see

MultiUpdate (p. 55), **Update** (p. 167)

Syntax

SetPWMFrequency *axis frequency*
GetPWMFrequency *axis*

Motor Types

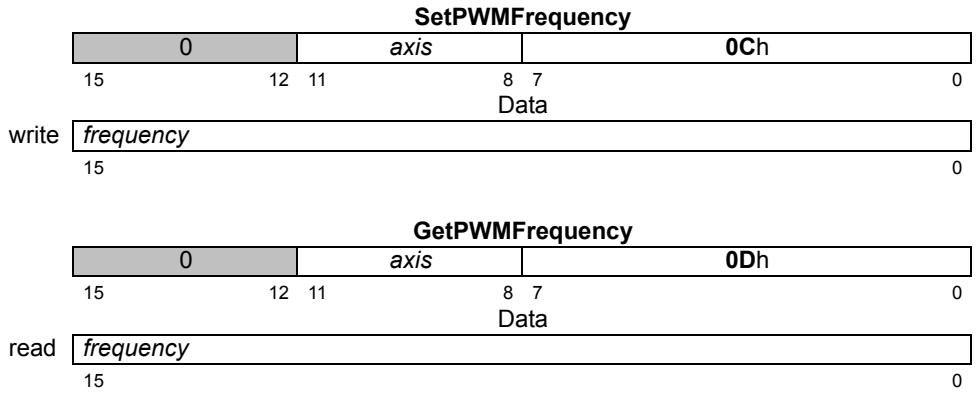
DC Brush	Brushless DC	Microstepping	

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

	Type	Range	Scaling	Units
frequency	unsigned 16 bits	0 *to* $2^{16}-1$	$1/2^8$	kHz

Packet Structure



Description

SetPWMFrequency sets the PWM output frequency (in kHz) for the specified ***axis***. To select one of the supported frequencies, pass the value listed in the SetPWMFrequency Value column as the ***frequency*** argument to this command.

Approximate Frequency	PWM bit Resolution	Actual Frequency	SetPWMFrequency Value
20 kHz	10	19.531 kHz	5,000
40 kHz	9	39.062 kHz	10,000
80 kHz	8	78.124 kHz	20,000

Restrictions

Only 20 kHz and 80 kHz are currently supported by the Magellan motion processor. Only 20 kHz and 40 kHz are supported in the ION products.

The PWM frequency can be changed only when motor output is disabled (e.g., immediately after power-up or reset).

C-Motion API

```
PMDresult PMDSetPWMFrequency(PMDAxisInterface axis_intf,
                             PMDuint16 frequency)
PMDresult PMDGetPWMFrequency(PMDAxisInterface axis_intf,
                             PMDuint16* frequency)
```

VB-Motion API

```
Dim frequency as Short
MagellanAxis.PWMFrequency = frequency
frequency = MagellanAxis.PWMFrequency
```

see

SetOutputMode (p. 124)

SetSampleTime 3Bh
GetSampleTime 3Ch

Syntax

SetSampleTime *time*
GetSampleTime

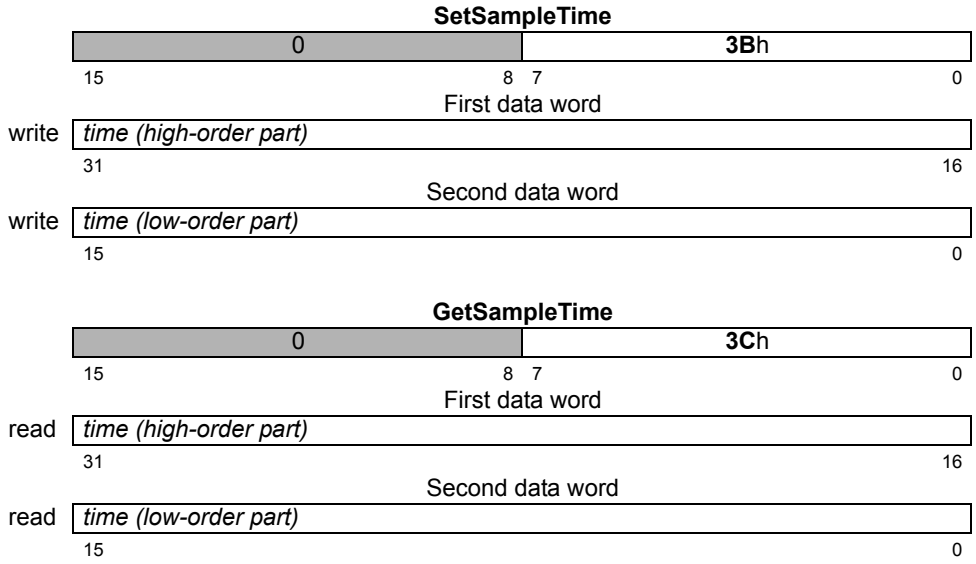
Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Type	Range	Units
time	unsigned 32 bits	51 *to* 2^{20}	microseconds

Packet Structure



Description

SetSampleTime sets the time basis for the motion processor. This time basis determines the trajectory update rate for all motor types as well as the servo loop calculation rate for DC brush and brushless DC motors. It does not, however, determine the commutation rate of the brushless DC motor types, nor the PWM or current loop rates for any motor type.

The ***time*** value is expressed in microseconds. The motion processor hardware can adjust the cycle time only in increments of 51.2 microseconds; the ***time*** value passed to this command will be rounded up to the nearest increment of this base value.

Minimum cycle time depends on the product and number of enabled axes as follows:

# Enabled Axes	Minimum Cycle Time	Cycle Time w/ Trace Capture	Time per Axis	Maximum Cycle Frequency
1 (ION)	102.4 µs	102.4 µs	102.4 µs	9.76 kHz
1 (Magellan Single-axis)	51.2 µs	102.4 µs	51.2 µs	19.53 kHz (9.76 w/ trace capture)
1 (Magellan Multi-axis)	102.4 µs	102.4 µs	102.4 µs	9.76 kHz
2 (Magellan)	153.6 µs	153.6 µs	76.8 µs	6.51 kHz
3 (Magellan)	204.8 µs	204.8 µs	68.3 µs	4.88 kHz
4 (Magellan)	256 µs	256 µs	64 µs	3.91 kHz

Description (cont.)

Using the trace feature on single axis Magellan products with the sample time set to 51.2 µs will result in unexpected behavior.

GetSampleTime returns the value of the sample time.

Restrictions

This command affects the cycle time for all axes on multi-axis configurations.

This command cannot be used to set a sample time lower than the required minimum cycle time for the current configuration. Attempting to do so will set the sample time to the required minimum cycle time as specified in the previous table.

C-Motion API

```
PMDresult PMDSetSampleTime(PMDAxisInterface axis_intf,
                           PMDuint32 time)
PMDresult PMDGetSampleTime(PMDAxisInterface axis_intf,
                           PMDuint32* time)
```

VB-Motion API

```
Dim time as Long
MagellanAxis.SampleTime = time
time = MagellanAxis.SampleTime
```

see

SetSerialPortMode 8Bh
GetSerialPortMode 8Ch

Syntax

SetSerialPortMode *mode*
GetSerialPortMode

Motor Types

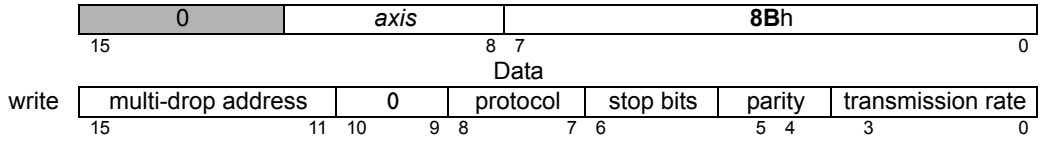
DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

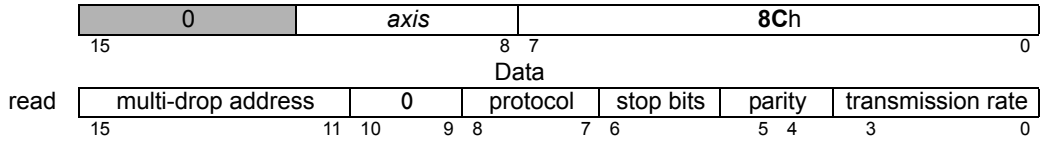
Name	Type	Encoding
mode	unsigned 16 bits	see below

Packet Structure

SetSerialPortMode



GetSerialPortMode



Description

SetSerialPortMode sets the configuration for the asynchronous serial port. It configures the timing and framing of the serial port on the unit, regardless of whether RS-232 or RS-485 voltage levels are being used. This command takes effect immediately, and the response will beat the new transmission rate, parity, etc.

GetSerialPortMode returns the configuration for the asynchronous serial port, regardless of whether RS-232 or RS-485 voltage levels are being used.

The following table shows the encoding of the data used by this command.

Bit Number	Name	Instance	Encoding
0–3	Transmission Rate	1200 baud	0
		2400 baud	1
		9600 baud	2
		19200 baud	3
		57600 baud	4
		115200 baud	5
		230400 baud	6
		460800 baud	7
4–5	Parity	none	0
		odd	1
		even	2
6	Stop Bits	1	0
		2	1
7–8	Protocol	Point-to-point	0
		Multi-drop using idle-line detection	1
		— (Reserved)	2
		— (Reserved)	3
11–15	Multi-Drop Address	Address 0	0
		Address 1	1
			...
		Address 31	31

Restrictions

C-Motion API

```
PMDresult PMDSetSerialPortMode(PMDAxisInterface axis_intf,
                               PMDuint8 baud,
                               PMDuint8 parity,
                               PMDuint8 stopBits,
                               PMDuint8 protocol,
                               PMDuint8 multiDropID)
PMDresult PMDGetSerialPortMode(PMDAxisInterface axis_intf,
                               PMDuint8* baud,
                               PMDuint8* parity,
                               PMDuint8* stopBits,
                               PMDuint8* protocol,
                               PMDuint8* multiDropID)
```

VB-Motion API

```
CommunicationSerial.SerialPortModeSet( [in] baud,
                                       [in] parity,
                                       [in] stopBits,
                                       [in] protocol,
                                       [in] multidropID )
```

see

SetSettleTime AAh
GetSettleTime ABh

Syntax

SetSettleTime *axis time*
GetSettleTime *axis*

Motor Types

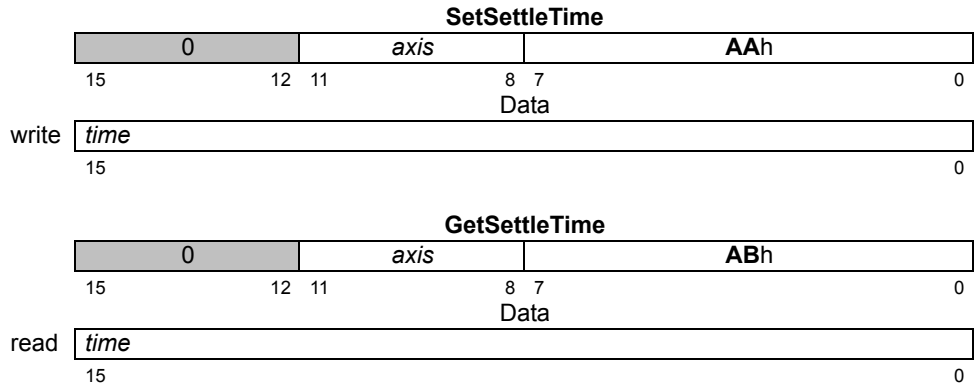
DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

	Type	Range	Scaling	Units
time	unsigned 16 bits	0 *to* $2^{15}-1$	unity	cycles

Packet Structure



Description

SetSettleTime sets the time, in number of cycles, that the specified ***axis*** must remain within the settle window before the Axis Settled indicator in the Activity Status register is set.

GetSettleTime returns the value of the settle time for the specified ***axis***.

Restrictions

C-Motion API

```
PMDresult PMDSetSettleTime(PMDAxisInterface axis_intf,
                           PMDuint16 time)
PMDresult PMDGetSettleTime(PMDAxisInterface axis_intf,
                           PMDuint16* time)
```

VB-Motion API

```
Dim time as Short
MagellanAxis.SettleTime = time
time = MagellanAxis.SettleTime
```

see

Set/GetMotionCompleteMode (p. 116), **Set/GetSettleWindow** (p. 145), **GetActivityStatus** (p. 26)

Syntax

SetSettleWindow *axis window*
GetSettleWindow *axis*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

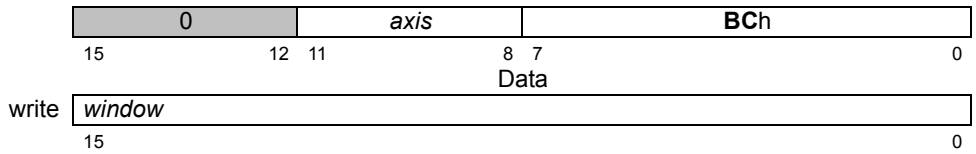
Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

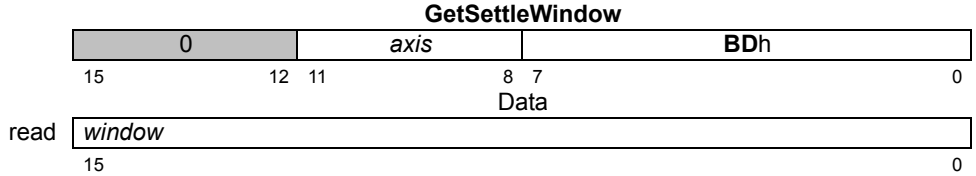
	Type	Range	Scaling	Units
window	unsigned 16 bits	0 *to* 2^{16}–1	unity	counts

Packet Structure

SetSettleWindow



GetSettleWindow



Description

SetSettleWindow sets the position range within which the specified ***axis*** must remain for the duration specified by **SetSettleTime** before the Axis Settled indicator in the Activity Status register is set.

GetSettleWindow returns the value of the settle window.

Restrictions

C-Motion API

```
PMDresult PMDSetSettleWindow(PMDAxisInterface axis_intf,
                             PMDuint16 window)
PMDresult PMDGetSettleWindow(PMDAxisInterface axis_intf,
                             PMDuint16* window)
```

VB-Motion API

```
Dim window as Short
MagellanAxis.SettleWindow = window
window = MagellanAxis.SettleWindow
```

see

Set/GetMotionCompleteMode (p. 116), **Set/GetSettleTime** (p. 144), **GetActivityStatus** (p. 26)

SetSignalSense A2h
GetSignalSense A3h

Syntax

SetSignalSense *axis sense*
GetSignalSense *axis*

Motor Types

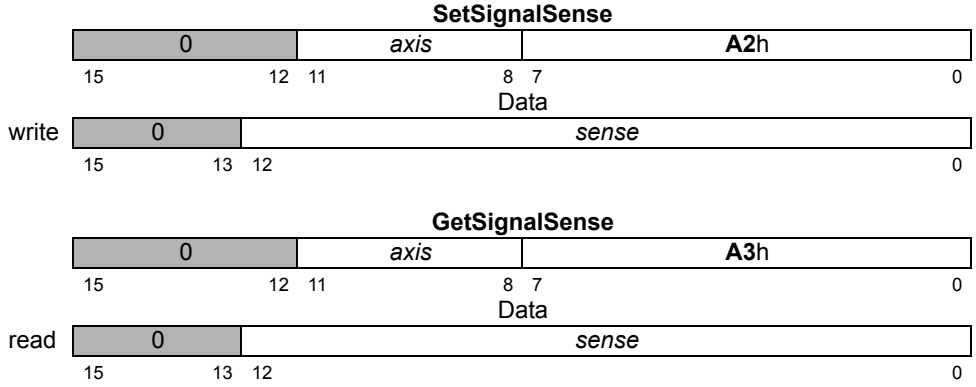
DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

	Indicator	Encoding	Bit Number
sense	*EncoderA*	0001h	0
	EncoderB	0002h	1
	Encoder Index	0004h	2
	Capture Input	0008h	3
	Positive Limit	0010h	4
	Negative Limit	0020h	5
	AxisIn	0040h	6
	HallA	0080h	7
	HallB	0100h	8
	HallC	0200h	9
	AxisOut	0400h	10
	Step Output	0800h	11
	Motor Direction	1000h	12
	— (Reserved)		13–15

Packet Structure



Description

SetSignalSense establishes the sense of the corresponding bits of the Signal Status register, with the addition of *Step Output* and *Motor Direction*, for the specified *axis*.

For *Encoder Index*, if the sense bit is 1, an index will be recognized for use in index-based phase correction if the index is high.

For the *Capture Input*, if the sense bit is 1, a capture will occur on a low-to-high signal transition. Otherwise, a capture will occur on a high-to-low transition.

For *Positive Limit* and *Negative Limit*: if the sense bit is 1, an overtravel condition will occur if the signal is high. Otherwise, an overtravel condition will occur when the signal is low.

Description (cont.)

The AxisOut signal is inverted if the sense bit is set to one; otherwise it is not inverted.

When the Step Output bit is set to 1, a step will be generated by the motion processor with a low-to-high transition on the Pulse signal. Otherwise, a step will be generated by the motion processor with a high-to-low transition on the Pulse signal.

Setting the Motor Direction bit has the effect of swapping the sense of positive and negative motor movement.

GetSignalSense returns the value of the Signal Sense mask.

Restrictions

In ION products, FaultOut and /Enable exist in the Signal Status register, but their sense is not controllable.

In ION products, when the Capture Source is set to Encoder Index, only the Encoder Index bit of signal sense should be used to configure its polarity. The Capture Input bit of Signal Sense should always be cleared to zero (0) in this case.

Not all bits are implemented for all products. See the product user's guide.

C-Motion API

```
PMDresult PMDSetSignalSense(PMDAxisInterface axis_intf,
                            PMDuint16 sense)
PMDresult PMDGetSignalSense(PMDAxisInterface axis_intf,
                            PMDuint16* sense)
```

VB-Motion API

```
Dim sense as Short
MagellanAxis.SignalSense = sense
sense = MagellanAxis.SignalSense
```

see

GetSignalStatus (p. 47)

SetSPIMode 0Ah
GetSPIMode 0Bh

Syntax

SetSPIMode *mode*
GetSPIMode

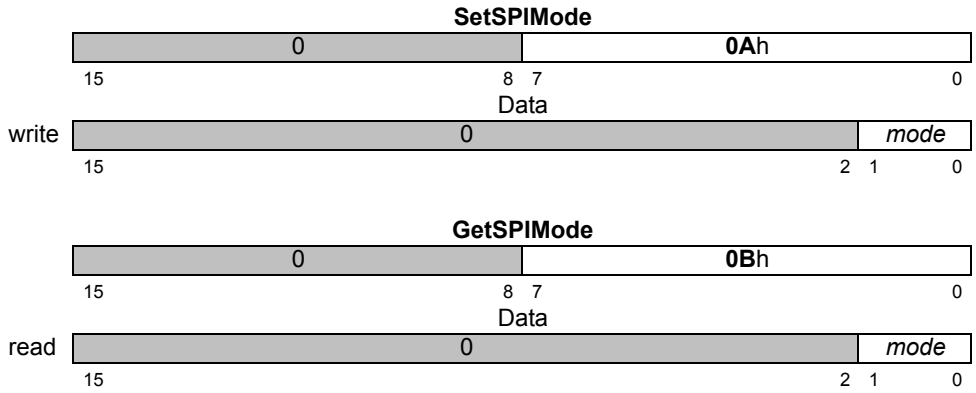
Motor Types

DC Brush			

Arguments

Name	Instance	Encoding
mode	*RisingEdge*	0
	RisingEdgeDelay	1
	FallingEdge	2
	FallingEdgeDelay	3

Packet Structure



Description

SetSPIMode configures the communication settings for the motion processor's SPI (Serial Peripheral Interface) DAC output port. Data is output as a series of 16-bit data words transmitted at 10 Mbps. The ***mode*** parameter controls the data clocking scheme as shown in the following table.

Mode	Encoding	Description
RisingEdge	0	Rising edge without phase delay: The SPIClock signal is inactive low. The SPIXmt pin transmits data on the rising edge of the SPIClock signal.
RisingEdgeDelay	1	Rising edge with phase delay: The SPIClock signal is inactive low. The SPIXmt pin transmits data one half-cycle ahead of the rising edge of the SPIClock signal.
FallingEdge	2	Falling edge without phase delay: The SPIClock signal is inactive high. The SPIXmt pin transmits data on the falling edge of the SPIClock signal.
FallingEdgeDelay	3	Falling edge with phase delay: The SPIClock signal is inactive high. The SPIXmt pin transmits data one half-cycle ahead of the falling edge of the SPIClock signal.

Restrictions

SPI output is only available when the motor type is **DC brush**, and only in some products. See the product user's guide.

C-Motion API

```
PMDresult PMDSetSPIMode(PMDAxisInterface axis_intf, PMDuint16 mode)
PMDresult PMDGetSPIMode(PMDAxisInterface axis_intf, PMDuint16* mode)
```

VB-Motion API

```
Dim mode as Short
MagellanObject.SPIMode = mode
mode = MagellanObject.SPIMode
```

see

SetOutputMode (p. 124)

Syntax

SetStartVelocity *axis velocity*
GetStartVelocity *axis*

Motor Types

		Microstepping	Pulse & Direction

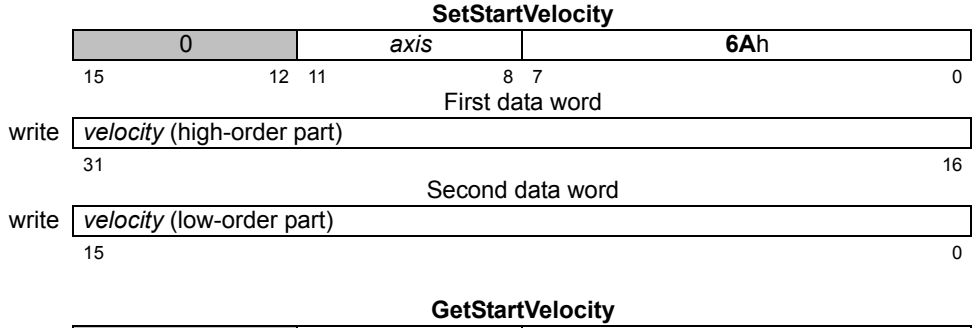
Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

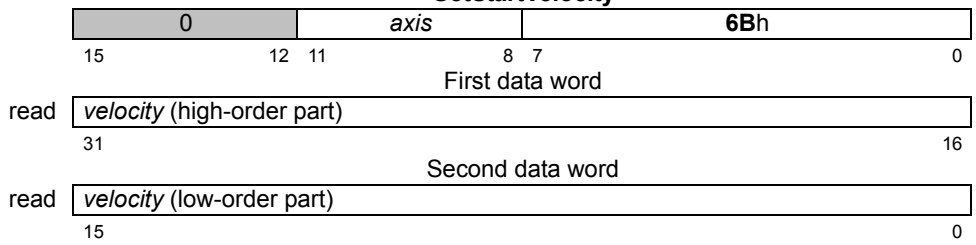
	Type	Range	Scaling	Units
velocity	unsigned 32 bits	0 *to* $2^{31}-1$	$1/2^{16}$	steps/cycle microsteps/cycle

Packet Structure

SetStartVelocity



GetStartVelocity



Description

SetStartVelocity loads the starting velocity register for the specified ***axis***. The start velocity is the instantaneous velocity at the start and at the end of the profile.

GetStartVelocity reads the value of the starting velocity register.

Scaling example: To load a starting velocity value of 1.750 steps/cycle multiply by 65,536 (giving 114,688) and load the resultant number as a 32-bit number, giving 0001 in the high word and C000h in the low word. Values returned by **GetStartVelocity** must correspondingly be divided by 65,536 to convert them to units of counts/cycle.

Restrictions

SetStartVelocity is only used in the Velocity Contouring and Trapezoidal profile modes.

C-Motion API

```
PMDresult PMDSetStartVelocity(PMDAxisInterface axis_intf,
                              PMDuint32 velocity)
PMDresult PMDGetStartVelocity(PMDAxisInterface axis_intf,
                              PMDuint32* velocity)
```

VB-Motion API

```
Dim velocity as Long
MagellanAxis.StartVelocity = velocity
velocity = MagellanAxis.StartVelocity
```

see

Set/GetVelocity (p. 165), **Set/GetAcceleration** (p. 66), **Set/GetDeceleration** (p. 95), **Set/GetPosition** (p. 133)

SetStepRange CFh
GetStepRange CEh

Syntax

SetStepRange *axis range*
GetStepRange *axis*

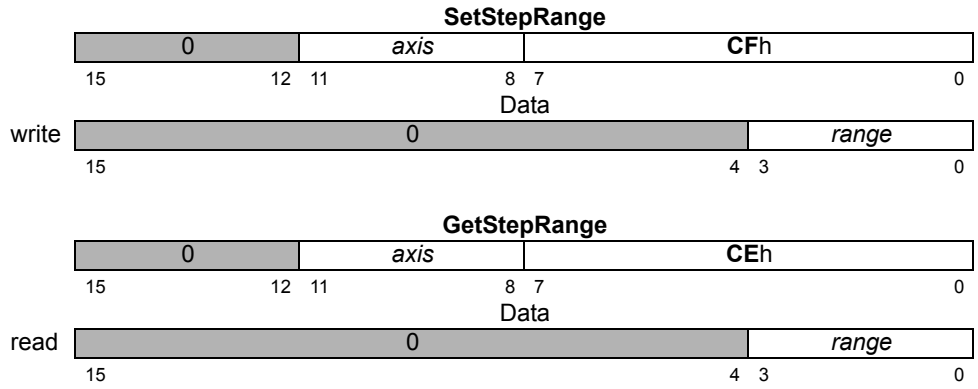
Motor Types

			Pulse & Direction

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
range	*0–4.98 Msteps/sec*	1
	0–622.5 ksteps/sec	4
	0–155.6 ksteps/sec	6
	0–38906 steps/sec	8

Packet Structure



Description

SetStepRange sets the maximum pulse rate frequency for the specified ***axis***. For example, if the desired maximum pulse rate is 200,000 pulses/second, the **SetStepRange 4** command should be issued.

GetStepRange returns the maximum pulse rate frequency for the specified ***axis***.

Restrictions

The MC55110 and the MC58110 have a maximum step range of 100 ksteps, which cannot be changed.

SetStepRange must be called before any moves are made, and must not be called after any moves have been made.

C-Motion API

```
PMDresult PMDSetStepRange(PMDAxisInterface axis_intf,
                          PMDuint16 range)
PMDresult PMDGetStepRange(PMDAxisInterface axis_intf,
                          PMDuint16* range)
```

VB-Motion API

```
Dim range as Short
MagellanAxis.StepRange = range
range = MagellanAxis.StepRange
```

see

Syntax

SetStopMode *axis mode*
GetStopMode *axis*

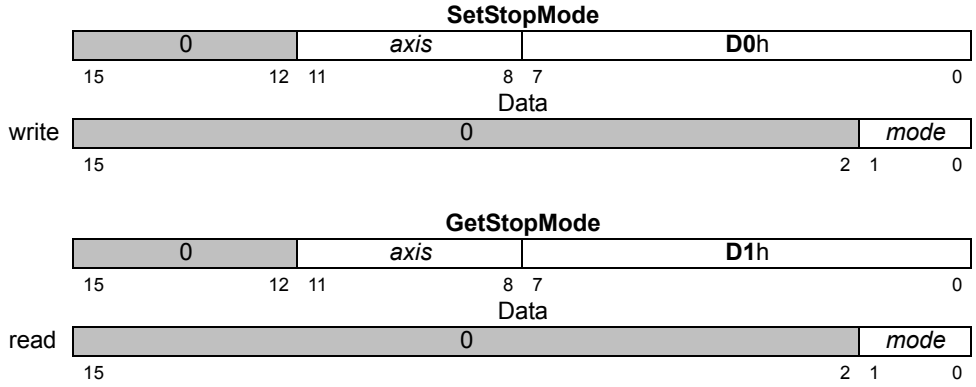
Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
mode	*No Stop*	0
	Abrupt Stop	1
	Smooth Stop	2

Packet Structure



Description

SetStopMode stops the specified *axis*. The available stop modes are *Abrupt Stop*, which instantly (without any deceleration phase) stops the axis; *Smooth Stop,* which uses the programmed deceleration value and profile shape for the current profile mode to stop the axis; or *No Stop*, which is generally used to turn off a previously issued set stop command.

Note: After an **Update**, a buffered stop command (**SetStopMode** command) will reset to the *No Stop* condition. In other words, if the **SetStopMode** command is followed by an **Update** command and then by a **GetStopMode** command, the retrieved stop mode will be *No Stop*.

GetStopMode returns the value of the stop mode.

Restrictions

Smooth Stop mode is not available in the Electronic Gear profile mode.

SetStopMode is a buffered command. The value set using this command will not take effect until the next **Update** or **MultiUpdate** command, with the Trajectory Update bit set in the update mask.

C-Motion API

```
PMDresult PMDSetStopMode(PMDAxisInterface axis_intf, PMDuint16 mode)
PMDresult PMDGetStopMode(PMDAxisInterface axis_intf, PMDuint16* mode)
```

VB-Motion API

```
Dim mode as Short
MagellanAxis.StopMode = mode
mode = MagellanAxis.StopMode
```

see

MultiUpdate (p. 55), **Update** (p. 167)

SetSynchronizationMode F2h
GetSynchronizationMode F3h

Syntax

SetSynchronizationMode *mode*
GetSynchronizationMode

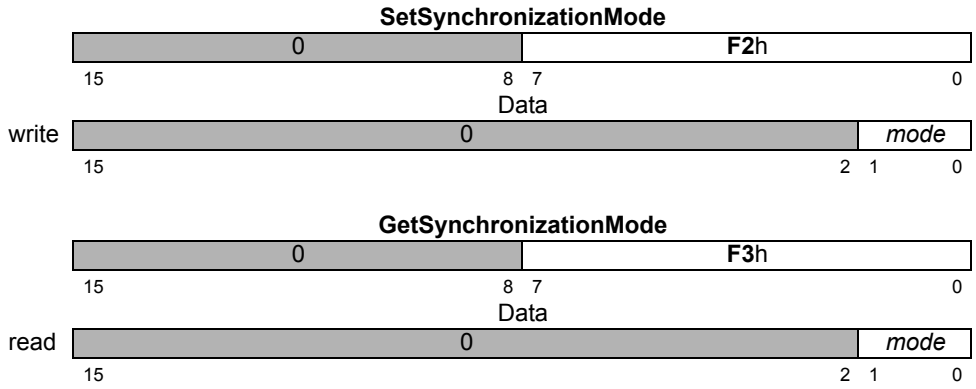
Motor Types

DC Brush	Brushless DC	Microstepping	

Arguments

Name	Instance	Encoding
mode	*Disabled*	0
	Master	1
	Slave	2

Packet Structure



Description

SetSynchronizationMode sets the mode of the pin used for the synchronization of the internal timer across multiple motion ICs. In the *Disabled* mode, the pin is configured as an input and is not used. In the *Master* mode, the pin outputs a synchronization pulse that can be used by slave nodes or other devices to synchronize with the internal chip cycle of the master node. In the *Slave* mode, the pin is configured as an input and a pulse on the pin synchronizes the internal chip cycle.

GetSynchronizationMode returns the value of the synchronization mode.

Restrictions

If the motion processor is configured as a slave, and any axis is configured for pulse & direction output, multi-chip synchronization cannot be used.

Multichip synchronization is not supported in all products. See the product user's guide.

C-Motion API

```
PMDresult PMDSetSynchronizationMode(PMDAxisInterface axis_intf,
                                    PMDuint16 mode)
PMDresult PMDGetSynchronizationMode(PMDAxisInterface axis_intf,
                                    PMDuint16* mode)
```

VB-Motion API

```
Dim mode as Short
MagellanObject.SynchronizationMode = mode
mode = MagellanObject.SynchronizationMode
```

see

Syntax

SetTraceMode *mode*
GetTraceMode

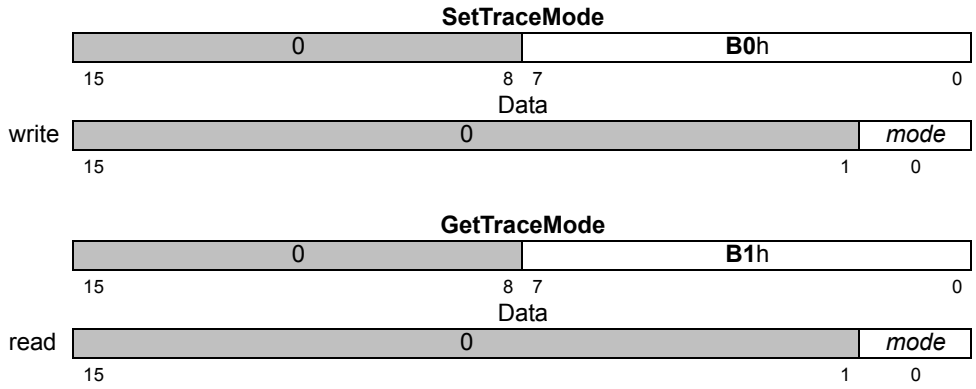
Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding
mode	*One Time*	0
	Rolling Buffer	1

Packet Structure



Description

SetTraceMode sets the buffer usage for the next trace. In *One Time* mode, the trace continues until the trace buffer is filled, then stops. In *Rolling Buffer* mode, the trace continues from the beginning of the trace buffer after the end is reached. When in the rolling mode, values stored at the beginning of the trace buffer are lost if they are not read before being overwritten by the wrapped data.

GetTraceMode returns the value for the trace mode.

Restrictions

C-Motion API

```
PMDresult PMDSetTraceMode(PMDAxisInterface axis_intf, PMDuint16 mode)
PMDresult PMDGetTraceMode(PMDAxisInterface axis_intf, PMDuint16* mode)
```

VB-Motion API

```
Dim mode as Short
MagellanObject.TraceMode = mode
mode = MagellanObject.TraceMode
```

see

GetTraceStatus (p. 51)

SetTracePeriod B8h
GetTracePeriod B9h

Syntax

SetTracePeriod *period*
GetTracePeriod

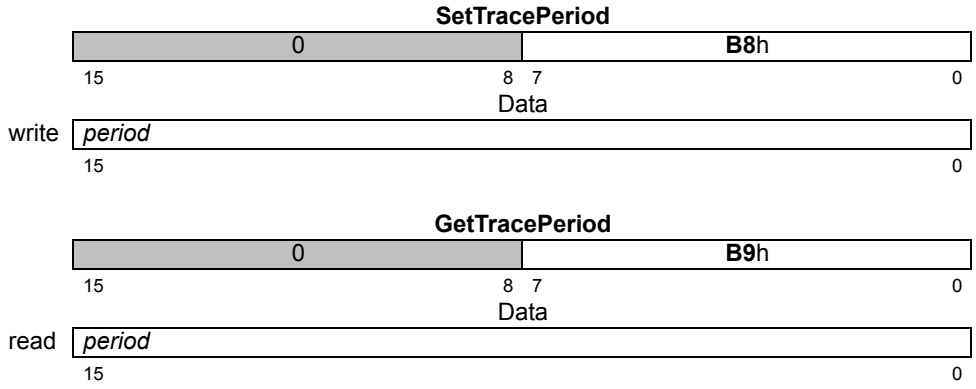
Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Type	Range	Scaling	Units
period	unsigned 16 bits	1 *to* $2^{16}-1$	unity	cycles

Packet Structure



Description

SetTracePeriod sets the interval between contiguous trace captures. For example, if the trace period is set to one, trace data will be captured at the end of every chip cycle. If the trace period is set to two, trace data will be captured at the end of every second chip cycle, and so on.

GetTracePeriod returns the value for the trace period.

Restrictions

C-Motion API

```
PMDresult PMDSetTracePeriod(PMDAxisInterface axis_intf,
                            PMDuint16 period);
PMDresult PMDGetTracePeriod(PMDAxisInterface axis_intf,
                            PMDuint16* period)
```

VB-Motion API

```
Dim period as Short
MagellanObject.TracePeriod = period
period = MagellanObject.TracePeriod
```

see

Set/GetSampleTime (p. 140), **Set/GetTraceStart** (p. 155), **Set/GetTraceStop** (p. 157)

Syntax

SetTraceStart *triggerAxis condition triggerBit triggerState*
GetTraceStart

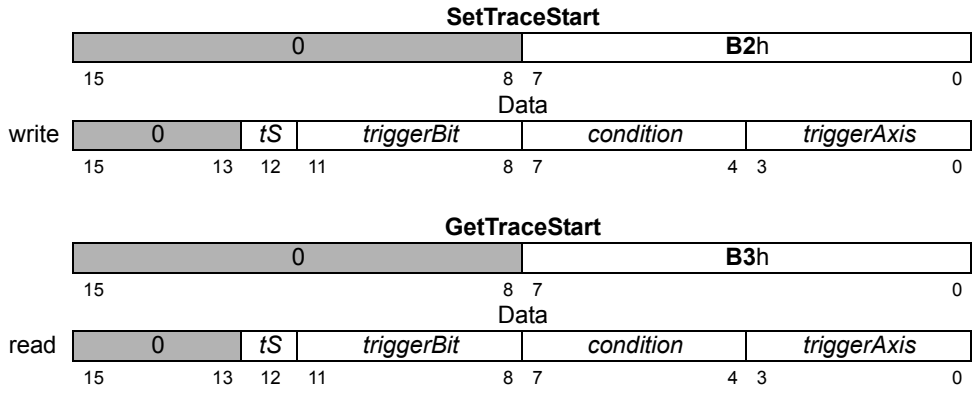
Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding
triggerAxis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
condition	*Immediate*	0
	Next Update	1
	Event Status	2
	Activity Status	3
	Signal Status	4
	Drive Status	5
triggerBit	*Status Register Bit*	0 *to* 15
triggerState (tS)	*Triggering State of the Bit*	0 (value = 0) 1 (value = 1)

Packet Structure



Description

SetTraceStart sets the condition for starting the trace. The ***Immediate*** condition requires no axis to be specified and the trace will begin upon execution of this instruction. The other four conditions require an axis to be specified, and when the condition for that axis is attained, the trace will begin.

When a status register bit is the trigger, the bit number and state must be included in the argument. The trace is started when the indicated bit reaches the specified state (0 or 1).

GetTraceStart returns the value of the trace-start trigger.

Once a trace has started, the trace-start trigger is reset to zero (0).

Description (cont.)

The following table shows the corresponding value for combinations of *triggerBit* and *register*0

TriggerBit	Event Status Register	Activity Status Register	Signal Status Register	Drive Status Register
0	Motion Complete	Phasing Initialized	Encoder A	
1	Wrap-around	At Maximum Velocity	Encoder B	In Foldback
2	Breakpoint 1	Tracking	Encoder Index	Overtemperature
3	Position Capture		Capture Input	
4	Motion Error		Positive Limit	In Holding
5	Positive Limit		Negative Limit	Overvoltage
6	Negative Limit		AxisIn	Undervoltage
7	Instruction Error	Axis Settled	Hall Sensor A	
8	Disable	Motor mode	Hall Sensor B	
9	Overtemperature Fault	Position Capture	Hall Sensor C	
0Ah	Bus Voltage Fault	In Motion		
0Bh	Commutation Error	In Positive Limit		
0Ch	Current Foldback	In Negative Limit		
0Dh			/Enable Input	
0Eh	Breakpoint 2		FaultOut	
0Fh				

Examples:

If it is desired that the trace begin on the next **Update** for axis 3, then a 2 is set for the axis number, a 1 is set for the condition, and bit number and state can be loaded with zeroes since they are not used. The actual data word sent to the motor processor in this case is 0012h.

If it is desired that the trace begin when bit 7 of the Activity Status register for axis 2 goes to 0, then the trace start is loaded as follows: A 1 is loaded for axis number, a 3 is loaded for condition, a 7 is loaded for bit number, and a 0 is loaded for state. The actual data word sent to the motor processor is 0731h.

Restrictions

Not all trace start conditions are available in all products. See the product user's guide.

C-Motion API

```
PMDresult PMDSetTraceStart(PMDAxisInterface axis_intf,
                           PMDAxis traceAxis,
                           PMDuint8 condition,
                           PMDuint8 triggerBit,
                           PMDuint8 triggerState)
PMDresult PMDGetTraceStart(PMDAxisInterface axis_intf,
                           PMDAxis* traceAxis,
                           PMDuint8* condition,
                           PMDuint8* triggerBit,
                           PMDuint8* triggerState)
```

VB-Motion API

```
MagellanObject.TraceStartSet( [in] triggerAxis,
                              [in] condition,
                              [in] triggerBit,
                              [in] triggerState )
MagellanObject.TraceStartGet( [out] triggerAxis,
                              [out] condition,
                              [out] triggerBit,
                              [out] triggerState )
```

see

Set/GetBufferLength (p. 81), **GetTraceCount** (p. 50), **Set/GetTraceMode** (p. 153), **Set/GetTracePeriod** (p. 154), **Set/GetTraceStop** (p. 157)

SetTraceStop B4h
GetTraceStop B5h

Syntax

SetTraceStop *triggerAxis condition triggerBit triggerState*
GetTraceStop

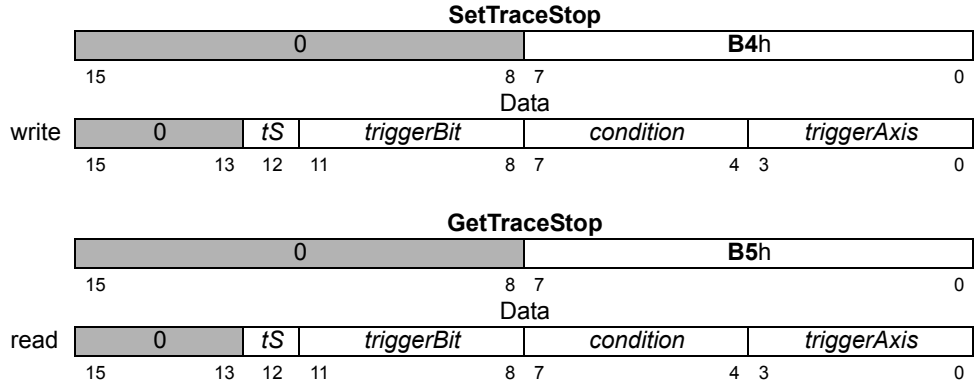
Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding
triggerAxis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
condition	*Immediate*	0
	Next Update	1
	Event Status	2
	Activity Status	3
	Signal Status	4
	Drive Status	5
triggerBit	*Status Register Bit*	0 *to* 15
triggerState (tS)	*Triggering State of the Bit*	0 (value = 0) 1 (value = 1)

Packet Structure



Description

SetTraceStop sets the condition for stopping the trace. The ***Immediate*** condition requires no axis to be specified and the trace will stop upon execution of this instruction. The other four conditions require an axis to be specified, and when the condition for that axis is attained, the trace will stop.

When a status register bit is the trigger, the bit number and state must be included in the argument. The trace is stopped when the indicated bit reaches the specified state (0 or 1).

GetTraceStop returns the value of the trace-stop trigger.

Once a trace has stopped, the trace-stop trigger is reset to zero (0).

Description (cont.)

The following table shows the corresponding value for combinations of *triggerBit* and *register*.

TriggerBit	Event Status Register	Activity Status Register	Signal Status Register	Drive Status Register
0	Motion Complete	Phasing Initialized	Encoder A	
1	Wrap-around	At Maximum Velocity	Encoder B	In Foldback
2	Breakpoint 1	Tracking	Encoder Index	Overtemperature
3	Position Capture		Capture Input	
4	Motion Error		Positive Limit	In Holding
5	Positive Limit		Negative Limit	Overvoltage
6	Negative Limit		AxisIn	Undervoltage
7	Instruction Error	Axis Settled	Hall Sensor A	
8	Disable	Motor mode	Hall Sensor B	
9	Overtemperature Fault	Position Capture	Hall Sensor C	
0Ah	Bus Voltage Fault	In Motion		
0Bh	Commutation Error	In Positive Limit		
0Ch	Current Foldback	In Negative Limit		
0Dh			/Enable Input	
0Eh	Breakpoint 2		FaultOut	
0Fh				

Examples:

If it is desired that the trace ends on the next **Update** for axis 3, then a 2 is set for the axis number, a 1 is set for the condition, and bit number and state can be loaded with zeroes since they are not used. The actual data word sent to the motor processor in this case is 0012h.

If it is desired that the trace ends when bit 7 of the Activity Status register for axis 2 goes to 0, then the trace stop is loaded as follows: A 1 is loaded for axis number, a 3 is loaded for condition, a 7 is loaded for bit number, and a 0 is loaded for state. The actual data word sent to the motor processor in this case is 0731h.

Restrictions

Not all trace stop conditions are available in all products. See the product user's guide.

C-Motion API

```
PMDresult PMDSetTraceStop(PMDAxisInterface axis_intf,
                          PMDAxis traceAxis,
                          PMDuint8 condition,
                          PMDuint8 triggerBit,
                          PMDuint8 triggerState)
PMDresult PMDGetTraceStop(PMDAxisInterface axis_intf,
                          PMDAxis* traceAxis,
                          PMDuint8* condition,
                          PMDuint8* triggerBit,
                          PMDuint8* triggerState)
```

VB-Motion API

```
MagellanObject.TraceStopSet( [in] triggerAxis,
                             [in] condition,
                             [in] triggerBit,
                             [in] triggerState )
MagellanObject.TraceStopGet( [out] triggerAxis,
                             [out] condition,
                             [out] triggerBit,
                             [out] triggerState )
```

see

GetTraceCount (p. 50), **Set/GetTraceStart** (p. 155), **GetTraceStatus** (p. 51)

SetTraceVariable B6h
GetTraceVariable B7h

Syntax

SetTraceVariable *variableNumber traceAxis variableID*
GetTraceVariable *variableNumber*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

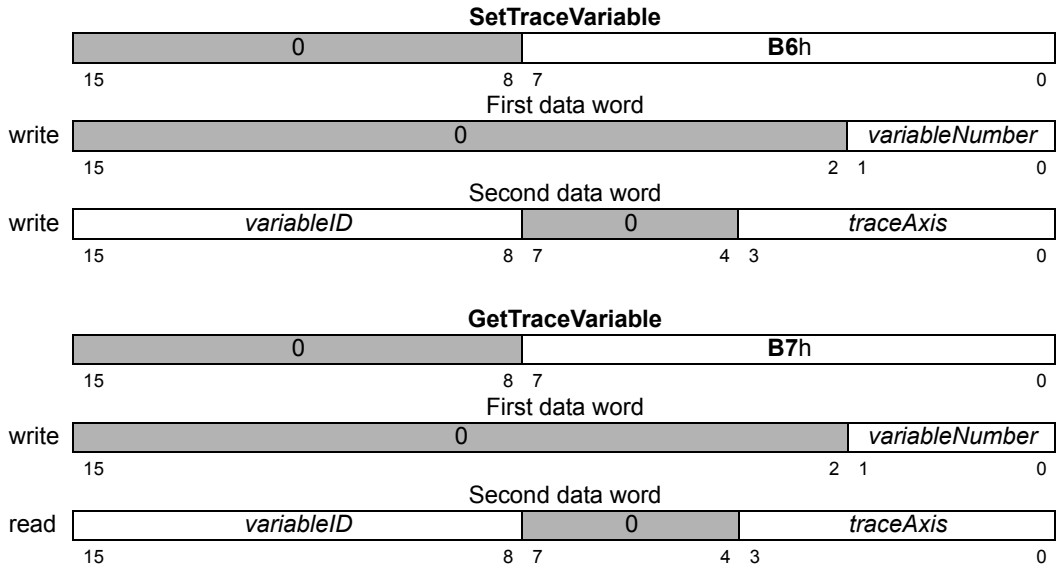
Arguments

Name	Instance	Encoding
variableNumber	*Variable1*	0
	Variable2	1
	Variable3	2
	Variable4	3
traceAxis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3
variableID		
Trajectory Generator	*Commanded Position*	02h
	Commanded Velocity	03h
	Commanded Acceleration	04h
Encoder	*Actual Position*	05h
	Actual Velocity	06h
	Capture Value	09h
	Phase Angle	0Fh
	Phase Offset	10h
Position Loop	*Position Error*	01h
	Position Loop Integral Sum	0Ah
	Position Loop Integral Contribution	39h
	Position Loop Derivative	0Bh
	PID Output (Biquad1 Input)	40h
	Biquad1 Output (Biquad2 Input)	41h
Status Registers	*Event Status Register*	0Ch
	Activity Status Register	0Dh
	Signal Status Register	0Eh
	Drive Status Register	38h
Commutation/Phasing	*Active Motor Command*	07h
	Phase A Command	11h
	Phase B Command	12h
	Phase C Command	13h
	Phase Angle Scaled	1Dh
Current Loops	*Phase A Reference*	42h
	Phase A Error	1Eh
	Phase A Actual Current	1Fh
	Phase A Integrator Sum	20h
	Phase A Integral Contribution	21h
	Current Loop A Output	22h
	Phase B Reference	43h
	Phase B Error	23h
	Phase B Actual Current	24h
	Phase B Integrator Sum	25h
	Phase B Integral Contribution	26h
	Current Loop B Output	27h
	D Feedback	2Ah
	Q Feedback	30h

Arguments (cont.)

Field Oriented Control	*D Reference*	28h
	D Error	29h
	D Feedback	2Ah
	D Integrator Sum	2Bh
	D Integral Contribution	2Ch
	D Output	2Dh
	Q Reference	2Eh
	Q Error	2Fh
	Q Feedback	30h
	Q Integrator Sum	31h
	Q Integral Contribution	32h
	Q Output	33h
	Phase A Actual Current	1Fh
	Phase B Actual Current	24h
	FOC Alpha Output	34h
	FOC Beta Output	35h
Motor Output	*Bus Voltage*	36h
	Temperature	37h
Analog Inputs	*Analog Input0*	14h
	Analog Input1	15h
	Analog Input2	16h
	Analog Input3	17h
	Analog Input4	18h
	Analog Input5	19h
	Analog Input6	1Ah
	Analog Input7	1Bh
Miscellaneous	*None* (disable variable)	00h
	Motion Processor Time	08h

Packet Structure



SetTraceVariable (cont.) B6h
GetTraceVariable B7h

Description

SetTraceVariable assigns the given variable to the specified *variableNumber* location in the trace buffer. Up to four variables may be traced at one time.

All variable assignments must be contiguous starting with *variableNumber* = 0.

GetTraceVariable returns the variable and axis of the specified *variableNumber*.

Example: To set up a three variable trace capturing the commanded acceleration for axis 1, the actual position for axis 1, and the event status word for axis 3, the following sequence of commands would be used. First, a **SetTraceVariable** command with *variableNumber* of 0, *axis* of 0, and *variableID* of 4 would be sent. Then, a **SetTraceVariable** command with *variableNumber* of 1, *axis* of 0, and *variableID* of 5 would be sent. Finally, a **SetTraceVariable** command with a *variableNumber* of 3, *axis* of 2 and *variableID* of 0h would be sent.

Restrictions

When selecting *ActualVelocity* as a trace variable, the reported value is the change in position between trace captures.

In FOC (Field Oriented Control) mode, A/B current values are not meaningful. Select D/Q current values instead.

Not all trace variables are available in all products. See the product user's guide.

C-Motion API

```
PMDresult PMDSetTraceVariable(PMDAxisInterface axis_intf,
                              PMDuint16 variableNumber,
                              PMDAxis traceAxis,
                              PMDuint8 variableID)
PMDresult PMDGetTraceVariable(PMDAxisInterface axis_intf,
                              PMDuint16 variableNumber,
                              PMDAxis* traceAxis,
                              PMDuint8* variableID)
```

VB-Motion API

```
MagellanObject.TraceVariableSet( [in] variableNumber,
                                 [in] traceAxis,
                                 [in] variableID )
MagellanObject.TraceVariableGet( [in] variableNumber,
                                 [out] traceAxis,
                                 [out] variableID )
```

see

SetTracePeriod (p. 154), **SetTraceStart** (p. 155), **SetTraceStop** (p. 157)

Syntax

SetTrackingWindow *axis window*
GetTrackingWindow *axis*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

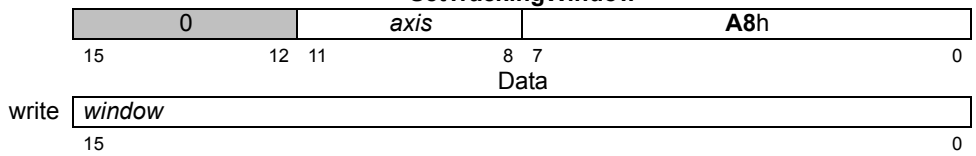
Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

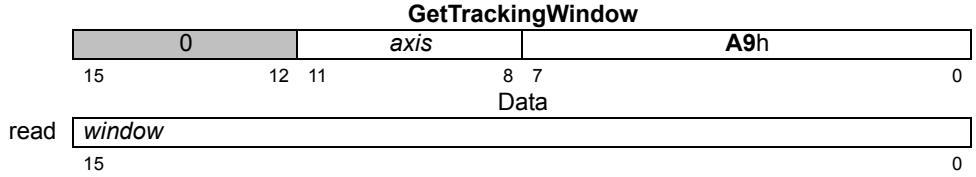
	Type	Range	Scaling	Units
window	unsigned 16 bits	0 *to* 2^{16}–1	unity	counts

Packet Structure

SetTrackingWindow



GetTrackingWindow



Description

SetTrackingWindow sets boundaries for the position error of the specified *axis*. If the absolute value of the position error exceeds the tracking window, the tracking indicator (bit 2 of the Activity Status register) is set to 0. When the position error returns to within the window, the tracking indicator is set to 1.

GetTrackingWindow returns the value of the tracking window.

Restrictions

C-Motion API

```
PMDresult PMDSetTrackingWindow(PMDAxisInterface axis_intf,
                               PMDuint16 window)
PMDresult PMDGetTrackingWindow(PMDAxisInterface axis_intf,
                               PMDuint16* window)
```

VB-Motion API

```
Dim window as Short
MagellanAxis.TrackingWindow = window
window = MagellanAxis.TrackingWindow
```

see

GetActivityStatus (p. 26), **GetActualPosition** (p. 68)

SetUpdateMask F9h
GetUpdateMask FAh

Syntax

SetUpdateMask *axis mask*
GetUpdateMask *axis*

Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

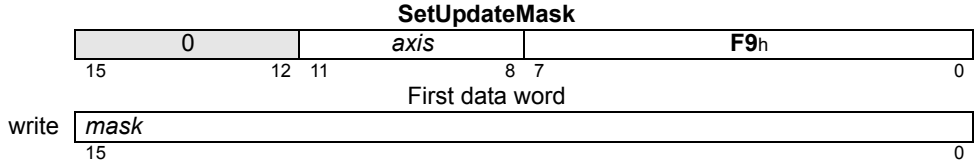
Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

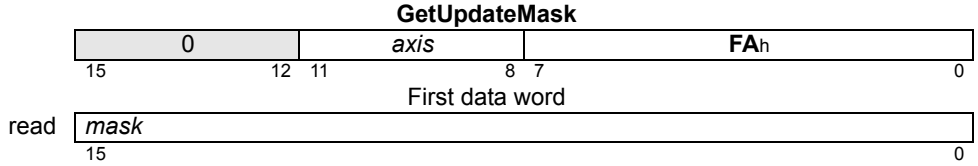
	Type	Scaling
mask	unsigned 16 bit	bitmask

Packet Structure

SetUpdateMask



GetUpdateMask



Description

SetUpdateMask configures what loops in the ***axis*** are updated when an update is executed on the given ***axis***. If the bitmask for a given loop is set in the ***mask***, the operating parameters for that loop will be updated from the buffered values when an **Update** or **MultiUpdate** command is received. The bitmask encoding is given below.

Name	Bit(s)	Description
Trajectory	0	Set to 1 to update trajectory from buffered parameters.
Position Loop	1	Set to 1 to update position loop from buffered parameters.
—	2	Reserved
Current Loop	3	Set to 1 to update current loop from buffered parameters.
—	4–15	Reserved

For example, if the update mask for a given ***axis*** is set to hexadecimal 0003h, the trajectory and position loop parameters will be updated from their buffered values when an **Update** or **MultiUpdate** command is received for that ***axis***.

The Current Loop bit applies regardless of the active current control mode. When it is set, an **Update** or **MultiUpdate** command will update either the active FOC parameters, or the active digital current loop parameters, depending on which Current Control mode is active.

GetUpdateMask gets the update mask for the indicated ***axis***.

Restrictions

The current loop bit is only valid for products that include a current loop.

C-Motion API

```
PMDresult PMDSetUpdateMask (PMDAxisInterface axis_intf,
                            PMDuint16 mask)
PMDresult PMDGetUpdateMask (PMDAxisInterface axis_intf,
                            PMDuint16* mask)
```

VB-Motion API

```
Dim mask as Short
MagellanAxis.UpdateMask = mask
mask = MagellanAxis.UpdateMask
```

see

Set/GetBreakpointUpdateMask (p. 163), **Update** (p. 167), **MultiUpdate** (p. 55)

SetVelocity buffered 11h
GetVelocity 4Bh

Syntax

SetVelocity *axis velocity*
GetVelocity *axis*

Motor Types

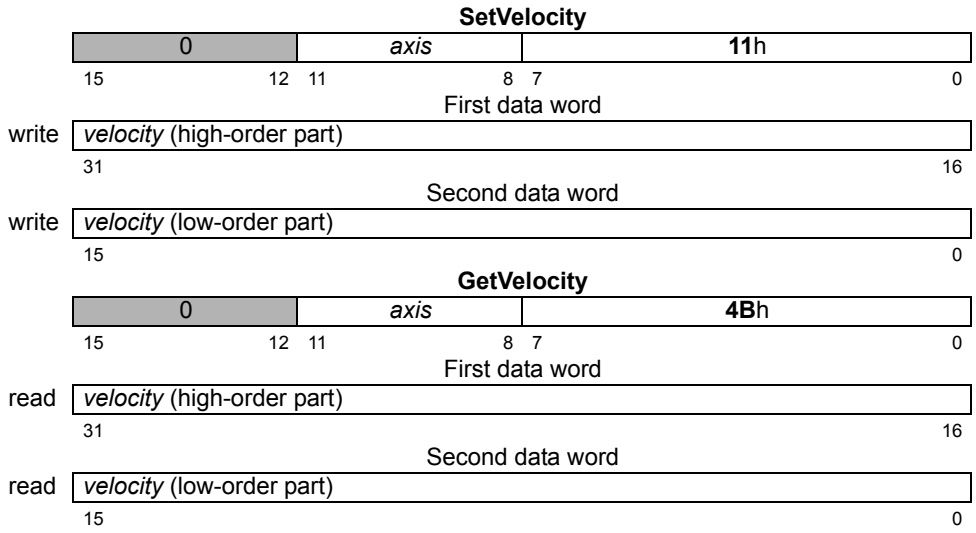
DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

	Type	Range	Scaling	Units
velocity	signed 32 bits	-2^{31} *to* $2^{31}-1$	$1/2^{16}$	counts/cycle microsteps/cycle

Packet Structure



Description

SetVelocity loads the maximum velocity buffer register for the specified ***axis***.

GetVelocity returns the contents of the maximum velocity buffer register.

Scaling example: To load a velocity value of 1.750 counts/cycle, multiply by 65,536 (giving 114,688) and load the resultant number as a 32-bit number; giving 0001 in the high word and C000h in the low word. Numbers returned by **GetVelocity** must correspondingly be divided by 65,536 to convert to units of counts/cycle.

Restrictions

SetVelocity may not be issued while an axis is in motion with the S-curve profile.

SetVelocity is not valid in Electronic Gear profile mode.

The velocity cannot be negative, except in the Velocity Contouring profile mode.

SetVelocity is a buffered command. The value set using this command will not take effect until the next **Update** or **MultiUpdate** command, with the Trajectory Update bit set in the update mask.

C-Motion API

```
PMDresult PMDSetVelocity(PMDAxisInterface axis_intf,
                         PMDint32 velocity)
PMDresult PMDGetVelocity(PMDAxisInterface axis_intf,
                         PMDint32* velocity)
```

VB-Motion API

```
Dim velocity as Long
MagellanAxis.Velocity = velocity
velocity = MagellanAxis.Velocity
```

see **Set/GetAcceleration** (p. 66), **Set/GetDeceleration** (p. 95), **Set/GetJerk** (p. 115), **Set/GetPosition** (p. 133), **MultiUpdate** (p. 55), **Update** (p. 167)

Syntax **Update** *axis*

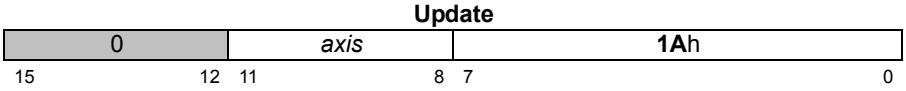
Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Instance	Encoding
axis	*Axis1*	0
	Axis2	1
	Axis3	2
	Axis4	3

Packet Structure



Description

Update causes all buffered data parameters to be copied into the corresponding run-time registers on the specified ***axis***. When the **Update** command is executed, the update mask is used to determine which groups of parameters are actually updated.

The following table shows the buffered commands and variables that are activated by the **Update** command.

Group	Command/Parameter
Trajectory	**Acceleration**
	Deceleration
	Gear Ratio
	Jerk
	Position
	Profile Mode
	Stop Mode
	Velocity
	Clear Position Error
Position Servo	**Derivative Time**
	Integration Limit
	Kaff
	Kd
	Ki
	Kp
	Kvff
	Kout
	Motor Command
Current Loops	**Integration Limit**
	Ki
	Kp

Restrictions

C-Motion API `PMDresult PMDUpdate(PMDAxisInterface axis_intf)`

VB-Motion API `MagellanAxis.Update()`

see **MultiUpdate** (p. 55), **Set/GetUpdateMask** (p. 163)

Syntax **WriteBuffer** *bufferID value*

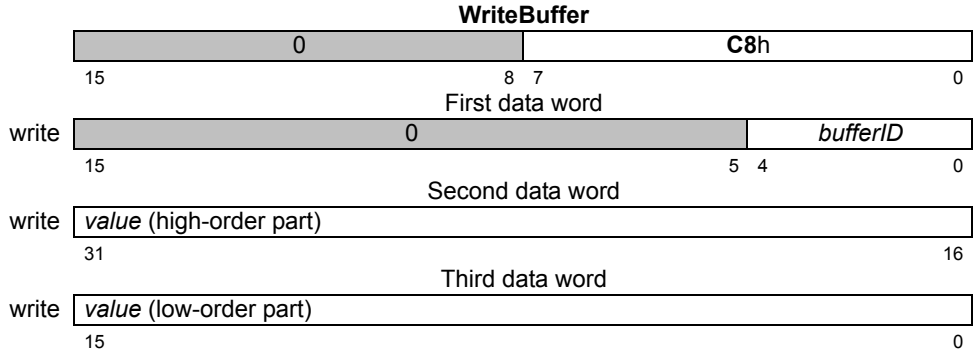
Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Type	Range
bufferID	unsigned 16 bits	0 *to* 31
value	signed 32 bits	-2^{31} *to* $2^{31}-1$

Packet Structure



Description **WriteBuffer** writes the 32-bit ***value*** into the location pointed to by the write buffer index in the specified buffer. After the contents have been written, the write index is incremented by 1. If the result is equal to the buffer length (set by **SetBufferLength**), the index is reset to zero (0).

Restrictions The command is not available on all products. See the product user's guide.

C-Motion API

```
PMDresult PMDWriteBuffer(PMDAxisInterface axis_intf,
                         PMDuint16 bufferID,
                         PMDint32 data)
```

VB-Motion API

```
Dim data as Long
MagellanObject.WriteBuffer( bufferID ) = data
```

see **ReadBuffer** (p. 58), **Set/GetBufferWriteIndex** (p. 84)

Syntax **WriteIO** *address data*

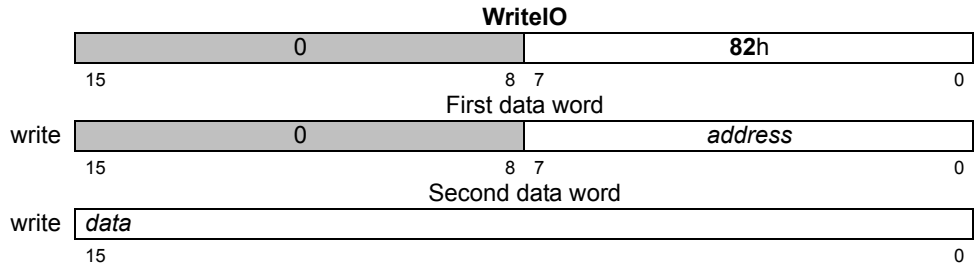
Motor Types

DC Brush	Brushless DC	Microstepping	Pulse & Direction

Arguments

Name	Type	Range
address	unsigned 16 bits	0 *to* 255
data	unsigned 16 bits	0 *to* 2^{16}–1

Packet Structure



Description

WriteIO writes one 16-bit word of ***data*** to ***address***. The ***address*** is an offset from location 1000h of the motion processor's peripheral device address space.

The format and interpretation of the 16-bit data word are dependent on the user-defined device being addressed. User-defined I/O can be used to implement a variety of features such as additional parallel I/O, flash memory for non-volatile configuration information storage, or display devices such as LED arrays.

Restrictions

This command is only available in products with general purpose parallel port interfaces. See the product user's guide.

C-Motion API

```
PMDresult PMDWriteIO(PMDAxisInterface axis_intf,
                     PMDuint16 address,
                     PMDuint16 data)
```

VB-Motion API

```
Dim data as Short
MagellanObject.IO( address ) = data
```

see **ReadIO** (p. 59)

This page intentionally left blank.

5. Instruction Summary Tables

5

5.1 Descriptions by Functional Category

Breakpoints and Interrupts		Page
Set/GetBreakpointUpdateMask	Set/Get mask for what is updated by breakpoint action "update."	77
ClearInterrupt	Reset interrupt.	22
Set/GetBreakpoint	Set/Get breakpoint type.	74
Set/GetBreakpointValue	Set/Get breakpoint comparison value.	79
GetInterruptAxis	Get the axes with pending interrupts.	43
Set/GetInterruptMask	Set/Get interrupt mask.	113

Motor Phase and Commutation		
Set/GetCommutationMode	Set/Get the commutation phasing mode.	89
Set/GetPhaseAngle	Set/Get current commutation phase angle.	126
GetPhaseCommand	Get the motor output command for a given phase A, B, or C.	44
Set/GetPhaseCorrectionMode	Set/Get phase correction mode.	127
Set/GetPhaseCounts	Set/Get number of encoder counts per commutation cycle.	128
Set/GetPhaseInitializeMode	Set/Get phase initialization method.	129
Set/GetPhaseInitializeTime	Set/Get the time parameters for algorithmic phase initialization.	130
Set/GetPhaseOffset	Set/Get phase offset value.	131
Set/GetPhasePrescale	Set/Get commutation prescaler mode.	132
InitializePhase	Perform phase initialization procedure.	54

Current Loops		
Set/GetCurrentControlMode	Set/Get current loop mode (PhaseA/B or FOC).	90
Set/GetCurrentLoop	Set/Get a parameter for the PhaseA/B current loops.	93
GetCurrentLoopValue	Get the instantaneous value of a node in the PhaseA/B current loops.	35
Set/GetFOC	Set/Get a parameter for the FOC current control.	106
GetFOCValue	Get the instantaneous value of a node in the FOC current control.	40

Digital Servo Filter		
ClearPositionError	Set position error to 0.	23
GetPositionError	Get actual position error.	45
Set/GetPositionLoop	Set/Get a parameter for the Digital Servo Loop.	135
GetPositionLoopValue	Get the current value of a node in the Digital Servo Loop.	46
Set/GetPositionErrorLimit	Set/Get the maximum position error limit.	134
Set/GetAuxiliaryEncoderSource	Controls the dual encoder loop feature.	71

Encoder		
AdjustActualPosition	Sums the specified offset with the actual encoder position.	20
Set/GetActualPosition	Set/Get the actual encoder position.	68
Set/GetActualPositionUnits	Set/Get the unit type returned for the actual encoder position.	70
GetActualVelocity	Get the actual encoder velocity.	28
Set/GetCaptureSource	Set/Get the capture source.	88
GetCaptureValue	Get current axis position capture value and reset the capture.	30

Encoder		
Set/GetEncoderModulus	Set/Get the full scale range of the parallel-word encoder.	98
Set/GetEncoderSource	Set/Get the encoder type.	99
Set/GetEncoderToStepRatio	Set/Get encoder count to step ratio.	101

Motor Output		
GetActiveMotorCommand	Read the active motor command value.	24
Set/GetMotorCommand	Set/Get direct value to motor output register, read buffered motor output command.	118
Set/GetMotorType	Set/Get motor type for axis.	120
Set/GetOutputMode	Set/Get the motor output mode.	124
Set/GetPWMFrequency	Set/Get the PWM output frequency	139
Set/GetStepRange	Set/Get the allowable range (in KHz) for step output generation.	150
Set/GetCurrentFoldback	Set/Get the maximum continuous operating current for I^2t Current Foldback	91
Set/GetHoldingCurrent	Set/Get parameters for holding current.	111
Set/GetMotorLimit	Set/Get motor output limit.	119
Set/GetMotorBias	Set/Get the motor bias (applied outside of position loop).	117

Operating Mode, Event, and Update Control		
Set/GetOperatingMode	Set/Get static Operating Mode of the axis.	122
RestoreOperatingMode	Restore the Current Operating Mode to the static Operating Mode.	65
GetActiveOperatingMode	Get the Active Operating Mode of the axis.	25
MultiUpdate	Forces buffered command values to become active for multiple axes.	55
Update	Forces buffered command values to become active.	167
Set/GetUpdateMask	Set/Get mask for what loops are updated by update command.	163
Set/GetEventAction	Set/Get the response of the axis to an event.	102

Postion Servo Loop Control		
Set/GetMotionCompleteMode	Set/Get the motion complete mode.	116
Set/GetSampleTime	Set/Get servo loop sample time.	140
Set/GetSettleTime	Set/Get the axis-settled time.	144
Set/GetSettleWindow	Set/Get the settle-window boundary value.	145
GetTime	Get current chipset time (number of servo loops).	49
Set/GetTrackingWindow	Set/Get the tracking window boundary value.	162

Profile Generation		
Set/GetAcceleration	Set/Get acceleration limit.	66
GetCommandedAcceleration	Get commanded (instantaneous desired) acceleration.	32
GetCommandedPosition	Get commanded (instantaneous desired) position.	33
GetCommandedVelocity	Get commanded (instantaneous desired) velocity.	34
Set/GetDeceleration	Set/Get deceleration limit.	95
Set/GetGearMaster	Set/Get the electronic gear mode master axis and source.	108
Set/GetGearRatio	Set/Get commanded electronic gear ratio.	110
Set/GetJerk	Set/Get jerk limit.	115
Set/GetPosition	Set/Get destination position.	133
Set/GetProfileMode	Set/Get current profile mode.	138
Set/GetStartVelocity	Set/Get start velocity.	149
Set/GetStopMode	Set/Get stop command: abrupt, smooth, or none.	151
Set/GetVelocity	Set/Get velocity limit.	165

RAM Buffer		
Set/GetBufferLength	Set/Get the length of a memory buffer.	81
Set/GetBufferReadIndex	Set/Get the buffer read pointer for a particular buffer.	82
Set/GetBufferStart	Set/Get the start location of a memory buffer.	83
Set/GetBufferWriteIndex	Set/Get the buffer write pointer for a particular buffer.	84
ReadBuffer	Read a long word value from a buffer memory location.	58
WriteBuffer	Write a long word value to a buffer memory location.	168

Drive		
Set/GetBusVoltageLimits	Set/Get threshold for Overvoltage or Undervoltage fault.	85
GetBusVoltage	Get the current bus voltage reading.	29
Set/GetOvertemperatureLimit	Set/Get threshold for Overtemperature fault.	125
GetTemperature	Gets current temperature reading.	48
Set/GetFaultOutMask	Set/Get mask for FaultOut from Event Status register.	104
GetDriveFaultStatus	Gets the Drive Fault Status register.	37
ClearDriveFaultStatus	Clears the Drive Fault Status register.	21

Status Registers and AxisOut Indicator		
GetActivityStatus	Get Activity Status register.	26
GetDriveStatus	Gets the Drive Status register.	38
Set/GetAxisOutMask	Set/Get AxisOut source.	72
GetEventStatus	Get Event Status word.	39
GetSignalStatus	Get the current axis Signal Status register.	47
Set/GetSignalSense	Set/Get the interpretation of the Signal Status bits.	146
ResetEventStatus	Reset bits in Event Status word.	64

Traces		
GetTraceCount	Get the number of traced data points.	50
Set/GetTraceMode	Set/Get the trace mode (rolling or one-time).	153
Set/GetTracePeriod	Set/Get the trace period.	154
Set/GetTraceStart	Set/Get the trace start condition.	155
GetTraceStatus	Get the trace status word.	51
Set/GetTraceStop	Set/Get the trace stop condition.	157
Set/GetTraceVariable	Set/Get a trace variable setting.	159

Communications		
Set/GetCANMode	Set/Get the CAN 2.0B configuration mode.	87
GetInstructionError	Get the most recent I/O error code.	42
Set/GetSerialPortMode	Set/Get the serial-port configuration mode.	142
Set/GetSPIMode	Set/Get the SPI output mode.	148

Miscellaneous		
GetChecksum	Reads the internal chip checksum.	31
Set/GetSynchronizationMode	Set/Get the synchronization mode.	152
GetVersion	Get chipset software version information.	52
NoOperation	Perform no operation, used to verify communications.	56
ReadIO	Read user-defined I/O value.	59
Reset	Reset chipset.	60
WriteIO	Write user-defined I/O value.	169
ReadAnalog	Read a raw analog input.	57
Set/GetDefault	Set/Get a reset default setting from non-volatile memory.	96

5.2 Alphabetical Listing

Get/Set instructions pairs are shown together on the same line of the table.

Instruction	Code	Instruction	Code	Page
AdjustActualPosition	F5h			20
ClearDriveFaultStatus	6Ch			21
ClearInterrupt	ACh			22
ClearPositionError	47h			23
GetAcceleration	4Ch	**SetAcceleration**	90h	66
GetActiveMotorCommand	3Ah			24
GetActiveOperatingMode	57h			25
GetActivityStatus	A6h			26
GetActualPosition	37h	**SetActualPosition**	4Dh	68
GetActualPositionUnits	BFh	**SetActualPositionUnits**	BEh	70
GetActualVelocity	ADh			28
GetAuxiliaryEncoderSource	09h	**SetAuxiliaryEncoderSource**	08h	71
GetAxisOutMask	46h	**SetAxisOutMask**	45h	72
GetBreakpoint	D5h	**SetBreakpoint**	D4h	74
GetBreakpointUpdateMask	33h	**SetBreakpointUpdateMask**	32h	77
GetBreakpointValue	D7h	**SetBreakpointValue**	D6h	79
GetBufferLength	C3h	**SetBufferLength**	C2h	81
GetBufferReadIndex	C7h	**SetBufferReadIndex**	C6h	82
GetBufferStart	C1h	**SetBufferStart**	C0h	83
GetBufferWriteIndex	C5h	**SetBufferWriteIndex**	C4h	84
GetBusVoltage	40h			29
GetBusVoltageLimits	60h	**SetBusVoltageLimits**	62h	85
GetCANMode	15h	**SetCANMode**	12h	87
GetCaptureSource	D9h	**SetCaptureSource**	D8h	88
GetCaptureValue	36h			30
GetChecksum	F8h			31
GetCommandedAcceleration	A7h			32
GetCommandedPosition	1Dh			33
GetCommandedVelocity	1Eh			34
GetCommutationMode	E3h	**SetCommutationMode**	E2h	89
GetCurrentControlMode	44h	**SetCurrentControlMode**	43h	90
GetCurrentFoldback	42h	**SetCurrentFoldback**	41h	91
GetCurrentLoop	74h	**SetCurrentLoop**	73h	93
GetCurrentLoopValue	71h			35
GetDeceleration	92h	**SetDeceleration**	91h	95
GetDefault	8Ah	**SetDefault**	89h	96
GetDriveFaultStatus	6Dh			37
GetDriveStatus	0Eh			38
GetEncoderModulus	8Eh	**SetEncoderModulus**	8Dh	98
GetEncoderSource	DBh	**SetEncoderSource**	DAh	99
GetEncoderToStepRatio	DFh	**SetEncoderToStepRatio**	DEh	101
GetEventAction	49h	**SetEventAction**	48h	102
GetEventStatus	31h			39

Instruction	Code	Instruction	Code	Page
GetFaultOutMask	FCh	**SetFaultOutMask**	FBh	104
GetFOC	F7h	**SetFOC**	F6h	106
GetFOCValue	5Ah			40
GetGearMaster	AFh	**SetGearMaster**	AEh	108
GetGearRatio	59h	**SetGearRatio**	14h	110
GetHoldingCurrent	5Fh	**SetHoldingCurrent**	5Eh	111
GetInstructionError	A5h			42
GetInterruptAxis	E1h			43
GetInterruptMask	56h	**SetInterruptMask**	2Fh	113
GetJerk	58h	**SetJerk**	13h	115
GetMotionCompleteMode	ECh	**SetMotionCompleteMode**	EBh	116
GetMotorBias	2Dh	**SetMotorBias**	0Fh	117
GetMotorCommand	69h	**SetMotorCommand**	77h	118
GetMotorLimit	07h	**SetMotorLimit**	06h	119
GetMotorType	03h	**SetMotorType**	02h	120
GetOperatingMode	66h	**SetOperatingMode**	65h	122
GetOutputMode	6Eh	**SetOutputMode**	E0h	124
GetOvertemperatureLimit	1Ch	**SetOvertemperatureLimit**	1Bh	125
GetPhaseAngle	2Ch	**SetPhaseAngle**	84h	126
GetPhaseCommand	EAh			44
GetPhaseCorrectionMode	E9h	**SetPhaseCorrectionMode**	E8h	127
GetPhaseCounts	7Dh	**SetPhaseCounts**	75h	128
GetPhaseInitializeMode	E5h	**SetPhaseInitializeMode**	E4h	129
GetPhaseInitializeTime	7Ch	**SetPhaseInitializeTime**	72h	130
GetPhaseOffset	7Bh	**SetPhaseOffset**	76h	131
GetPhasePrescale	E7h	**SetPhasePrescale**	E6h	132
GetPosition	4Ah	**SetPosition**	10h	133
GetPositionError	99h			45
GetPositionErrorLimit	98h	**SetPositionErrorLimit**	97h	134
GetPositionLoop	68h	**SetPositionLoop**	67h	135
GetPositionLoopValue	55h			46
GetProfileMode	A1h	**SetProfileMode**	A0h	138
GetPWMFrequency	0Dh	**SetPWMFrequency**	0Ch	139
GetSampleTime	3Ch	**SetSampleTime**	3Bh	140
GetSerialPortMode	8Ch	**SetSerialPortMode**	8Bh	142
GetSettleTime	ABh	**SetSettleTime**	AAh	144
GetSettleWindow	BDh	**SetSettleWindow**	BCh	145
GetSignalSense	A3h	**SetSignalSense**	A2h	146
GetSignalStatus	A4h			47
GetSPIMode	0Bh	**SetSPIMode**	0Ah	148
GetStartVelocity	6Bh	**SetStartVelocity**	6Ah	149
GetStepRange	CEh	**SetStepRange**	CFh	150
GetStopMode	D1h	**SetStopMode**	D0h	151
GetSynchronizationMode	F3h	**SetSynchronizationMode**	F2h	152
GetTemperature	53h			48
GetTime	3Eh			49
GetTraceCount	BBh			50
GetTraceMode	B1h	**SetTraceMode**	B0h	153
GetTracePeriod	B9h	**SetTracePeriod**	B8h	154
GetTraceStart	B3h	**SetTraceStart**	B2h	155
GetTraceStatus	BAh			51

Instruction	Code	Instruction	Code	Page
GetTraceStop	B5h	**SetTraceStop**	B4h	157
GetTraceVariable	B7h	**SetTraceVariable**	B6h	159
GetTrackingWindow	A9h	**SetTrackingWindow**	A8h	162
GetUpdateMask	FAh	**SetUpdateMask**	F9h	163
GetVelocity	4Bh	**SetVelocity**	11h	165
GetVersion	8Fh			52
InitializePhase	7Ah			54
MultiUpdate	5Bh			55
NoOperation	00h			56
ReadAnalog	EFh			57
ReadBuffer	C9h			58
ReadIO	83h			59
Reset	39h			60
ResetEventStatus	34h			64
RestoreOperatingMode	2Eh			65
Update	1Ah			167
WriteBuffer	C8h			168
WriteIO	82h			169

5.3 Numerical Listing

Code	Instruction	Page	Code	Instruction	Page
00h	**NoOperation**	56	48h	**SetEventAction**	102
02h	**SetMotorType**	120	49h	**GetEventAction**	102
03h	**GetMotorType**	120	4Ah	**GetPosition**	133
06h	**SetMotorLimit**	119	4Bh	**GetVelocity**	165
07h	**GetMotorLimit**	119	4Ch	**GetAcceleration**	66
08h	**SetAuxiliaryEncoderSource**	71	4Dh	**SetActualPosition**	68
09h	**GetAuxiliaryEncoderSource**	71	53h	**GetTemperature**	48
0Ah	**SetSPIMode**	148	55h	**GetPositionLoopValue**	46
0Bh	**GetSPIMode**	148	56h	**GetInterruptMask**	113
0Ch	**SetPWMFrequency**	139	57h	**GetActiveOperatingMode**	25
0Dh	**GetPWMFrequency**	139	58h	**GetJerk**	115
0Eh	**GetDriveStatus**	38	59h	**GetGearRatio**	110
0Fh	**SetMotorBias**	117	5Ah	**GetFOCValue**	40
10h	**SetPosition**	133	5Bh	**MultiUpdate**	55
11h	**SetVelocity**	165	5Eh	**SetHoldingCurrent**	111
12h	**SetCANMode**	87	5Fh	**GetHoldingCurrent**	111
13h	**SetJerk**	115	60h	**GetBusVoltageLimits**	85
14h	**SetGearRatio**	110	62h	**SetBusVoltageLimits**	85
15h	**GetCANMode**	87	65h	**SetOperatingMode**	122
1Ah	**Update**	167	66h	**GetOperatingMode**	122
1Bh	**SetOvertemperatureLimit**	125	67h	**SetPositionLoop**	135
1Ch	**GetOvertemperatureLimit**	125	68h	**GetPositionLoop**	135
1Dh	**GetCommandedPosition**	33	69h	**GetMotorCommand**	118
1Eh	**GetCommandedVelocity**	34	6Ah	**SetStartVelocity**	149
2Ch	**GetPhaseAngle**	126	6Bh	**GetStartVelocity**	149
2Dh	**GetMotorBias**	117	6Ch	**ClearDriveFaultStatus**	21
2Eh	**RestoreOperatingMode**	65	6Dh	**GetDriveFaultStatus**	37
2Fh	**SetInterruptMask**	113	6Eh	**GetOutputMode**	124
31h	**GetEventStatus**	39	71h	**GetCurrentLoopValue**	35
32h	**SetBreakpointUpdateMask**	77	72h	**SetPhaseInitializeTime**	130
33h	**GetBreakpointUpdateMask**	77	73h	**SetCurrentLoop**	93
34h	**ResetEventStatus**	64	74h	**GetCurrentLoop**	93
36h	**GetCaptureValue**	30	75h	**SetPhaseCounts**	128
37h	**GetActualPosition**	68	76h	**SetPhaseOffset**	131
39h	**Reset**	60	77h	**SetMotorCommand**	118
3Ah	**GetActiveMotorCommand**	24	7Ah	**InitializePhase**	54
3Bh	**SetSampleTime**	140	7Bh	**GetPhaseOffset**	131
3Ch	**GetSampleTime**	140	7Ch	**GetPhaseInitializeTime**	130
3Eh	**GetTime**	49	7Dh	**GetPhaseCounts**	128
40h	**GetBusVoltage**	29	82h	**WriteIO**	169
41h	**SetCurrentFoldback**	91	83h	**ReadIO**	59
42h	**GetCurrentFoldback**	91	84h	**SetPhaseAngle**	126
43h	**SetCurrentControlMode**	90	89h	**SetDefault**	96
44h	**GetCurrentControlMode**	90	8Ah	**GetDefault**	96
45h	**SetAxisOutMask**	72	8Bh	**SetSerialPortMode**	142
46h	**GetAxisOutMask**	72	8Ch	**GetSerialPortMode**	142
47h	**ClearPositionError**	23	8Dh	**SetEncoderModulus**	98
			8Eh	**GetEncoderModulus**	98
			8Fh	**GetVersion**	52

Code	Instruction	Page	Code	Instruction	Page
90h	**SetAcceleration**	66	D1h	**GetStopMode**	151
91h	**SetDeceleration**	95	D4h	**SetBreakpoint**	74
92h	**GetDeceleration**	95	D5h	**GetBreakpoint**	74
97h	**SetPositionErrorLimit**	134	D6h	**SetBreakpointValue**	79
98h	**GetPositionErrorLimit**	134	D7h	**GetBreakpointValue**	79
99h	**GetPositionError**	45	D8h	**SetCaptureSource**	88
A0h	**SetProfileMode**	138	D9h	**GetCaptureSource**	88
A1h	**GetProfileMode**	138	DAh	**SetEncoderSource**	99
A2h	**SetSignalSense**	146	DBh	**GetEncoderSource**	99
A3h	**GetSignalSense**	146	DEh	**SetEncoderToStepRatio**	101
A4h	**GetSignalStatus**	47	DFh	**GetEncoderToStepRatio**	101
A5h	**GetInstructionError**	42	E0h	**SetOutputMode**	124
A6h	**GetActivityStatus**	26	E1h	**GetInterruptAxis**	43
A7h	**GetCommandedAcceleration**	32	E2h	**SetCommutationMode**	89
A8h	**SetTrackingWindow**	162	E3h	**GetCommutationMode**	89
A9h	**GetTrackingWindow**	162	E4h	**SetPhaseInitializeMode**	129
AAh	**SetSettleTime**	144	E5h	**GetPhaseInitializeMode**	129
ABh	**GetSettleTime**	144	E6h	**SetPhasePrescale**	132
ACh	**ClearInterrupt**	22	E7h	**GetPhasePrescale**	132
ADh	**GetActualVelocity**	28	E8h	**SetPhaseCorrectionMode**	127
AEh	**SetGearMaster**	108	E9h	**GetPhaseCorrectionMode**	127
AFh	**GetGearMaster**	108	EAh	**GetPhaseCommand**	44
B0h	**SetTraceMode**	153	EBh	**SetMotionCompleteMode**	116
B1h	**GetTraceMode**	153	ECh	**GetMotionCompleteMode**	116
B2h	**SetTraceStart**	155	EFh	**ReadAnalog**	57
B3h	**GetTraceStart**	155	F2h	**SetSynchronizationMode**	152
B4h	**SetTraceStop**	157	F3h	**GetSynchronizationMode**	152
B5h	**GetTraceStop**	157	F5h	**AdjustActualPosition**	20
B6h	**SetTraceVariable**	159	F6h	**SetFOC**	106
B7h	**GetTraceVariable**	159	F7h	**GetFOC**	106
B8h	**SetTracePeriod**	154	F8h	**GetChecksum**	31
B9h	**GetTracePeriod**	154	F9h	**SetUpdateMask**	163
BAh	**GetTraceStatus**	51	FAh	**GetUpdateMask**	163
BBh	**GetTraceCount**	50	FBh	**SetFaultOutMask**	104
BCh	**SetSettleWindow**	145	FCh	**GetFaultOutMask**	104
BDh	**GetSettleWindow**	145			
BEh	**SetActualPositionUnits**	70			
BFh	**GetActualPositionUnits**	70			
C0h	**SetBufferStart**	83			
C1h	**GetBufferStart**	83			
C2h	**SetBufferLength**	81			
C3h	**GetBufferLength**	81			
C4h	**SetBufferWriteIndex**	84			
C5h	**GetBufferWriteIndex**	84			
C6h	**SetBufferReadIndex**	82			
C7h	**GetBufferReadIndex**	82			
C8h	**WriteBuffer**	168			
C9h	**ReadBuffer**	58			
CEh	**GetStepRange**	150			
CFh	**SetStepRange**	150			
D0h	**SetStopMode**	151			

5.4 Magellan Compatibility

Below are commands from Magellan v1.x that have been replaced/superseded by new commands in Magellan v2.x.

Old Command	Old Code	New Command
Set/GetBiquadCoefficient	04h/05h	**Set/GetPositionLoop**
GetDerivative	9Bh	**GetPositionLoopValue**
Set/GetDerivativeTime	9Ch/9Dh	**Set/GetPositionLoop**
GetHostIOError	A5h	**GetInstructionError (name change only)**
GetIntegral	9Ah	**GetPositionLoopValue**
Set/GetIntegrationLimit	95h/96h	**Set/GetPositionLoop**
Set/GetKaff	93h/94h	**Set/GetPositionLoop**
Set/GetKd	27h/52h	**Set/GetPositionLoop**
Set/GetKi	26h/51h	**Set/GetPositionLoop**
Set/GetKout	9Eh/9Fh	**Set/GetPositionLoop**
Set/GetKp	25h/50h	**Set/GetPositionLoop**
Set/GetKvff	2Bh/54h	**Set/GetPositionLoop**
Set/GetAutoStopMode	D2h/D3h	**Set/GetEventAction**
Set/GetMotorMode	DCh/DDh	**Set/GetOperatingMode, RestoreOperatingMode**
Set/GetAxisOutSource	EDh/EEh	**Set/GetAxisOutMask**
Set/GetAxisMode	87h/88h	**Set/GetOperatingMode**
Set/GetLimitSwitchMode	80h/81h	**Set/GetEventAction**

This page intentionally left blank.

For additional information, or for technical assistance, please contact PMD at (781) 674-9860.

You may also e-mail your request to support@pmdcorp.com

Visit our website at http://www.pmdcorp.com

Performance Motion Devices
55 Old Bedford Rd.
Lincoln, MA 01773